# SPIder Koerier

Juni 2007 Nummer 2 www.st-SPIder.nl


## Redactioneel

Met genoegen presenteren we de tweede Koerier van 2007. Met veel aandacht voor de SPIder conferentie op 2 oktober aanstaande. Het volledige programma vind je verderop in de Koerier.
Daarnaast een aantal actuele artikelen over o.a. *Software Product Release Management* met daarin een aantal zeer praktische handreikingen voor implementatie en een fraaie opsomming van misvattingen over dit onderwerp. Twee terugblikken op het Q-society events van 6 december 2006 en 9 mei jongstleden en een stukje over een succesvolle SPI-implementatie bij een ontwikkelunit bij Ericsson in Zweden.
Verder een artikel gaat kort in op Scrum, een aanpak uit de Agile-hoek.
Veel plezier bij het lezen ervan. We zien je feedback en natuurlijk ook je eigen artikelen met belangstelling tegemoet.

Mocht je nog een mededeling, suggestie of een artikel hebben waarvan je denkt dat het interessant zou kunnen zijn voor de SPIder leden, mail dan naar: koerier@st-SPIder.nl.

## Inhoudsopgave


- Redactioneel ..... 1
- Inhoudsopgave ..... 1
- Van het Bestuur ..... 1
- Making Software Process Improvement Happen ..... 2
- Data voor de 10e SPIder conferentie: ..... 7
- Infotentie ..... 8
- SPIder conferentie ..... 8
- SPIder en LinkedIn ..... 9
- Evaluatie 6 december 2006 Q workshop ..... 9
- Ten Misconceptions about Product Software Release Management explained using Cost/Value Functions ..... 11
- Scrum werkt… En soms ook minder… ..... 17
- Terugblik Q conferentie 9 mei 2007 ..... 20
- Nieuwsberichten & evenementenkalender .... 23
- Deelname in SPIder ..... 23
- Colofon ..... 23


## Van het Bestuur

Bij SPI denkt men vaak aan processen. Immers, de term staat voor Software Process Improvement. Maar uiteindelijk is het doel van SPI om de resultaten van de organisatie te verbeteren. Processen kunnen daar zeker bij helpen, maar er is nog een andere belangrijke factor: De Mens. Deze factor staat centraal in de SPIder conferentie op 2 oktober, met het thema "**Energyzing Improvements**".

Op de conferentie kijken we hoe professionals willen en kunnen veranderen, met de bijbehorende rol van de change agent. Opleiden en coachen spelen daarin een rol, maar ook het creëren van de juiste cultuur en andere randvoorwaarden. Ook belichten we het mens perspectief in nieuwe ontwikkelingen zoals Agile en Six Sigma. De presentaties komen uit de praktijk, en laten zien wat gewerkt heeft, wat niet, en waarom. Kortom, 2 oktober is een dag die je niet mag missen.

Vroege aanmelders (voor 15 juni) krijgen €50,- korting op de conferentie. Als je dan ook nog donateur bent of bij een van onze sponsors werkt, en betaalt per creditcard, kost de conferentie nog maar €249,-. Waar haal je nog zoveel kennis en ervaring, voor zo weinig geld? En, het wordt ook nog de 10e conferentie. Ik zal nog niets verraden, maar we gaan er zeker een feestje van maken…

Oké, tot zover de "reclame", nu even naar de praktijk van alledag. Als voorzitter mag ik de activiteiten van SPIder "verkopen", sterker nog, dat verwacht men van mij. Maar eigenlijk zijn we allemaal verkopers! Als je een organisatie wilt veranderen, heb je "buy-in" nodig. Je zult je ideeën moeten verkopen, aan de leiding van het bedrijf en ook aan de uitvoerende professionals. Hoe doe je dat? Gouden regel is om te laten zien welk voordeel er voor hun inzit, "what's in it for me?". Het lijkt zo basaal, en toch is het iets wat nog niet zo goed gaat. Verderop in de Koerier een artikel wat inzoomt

op deze problematiek onder de titel “Make SPI Happen”. Food for thought…

Terug naar de reclame. SPIder is dit jaar wederom betrokken bij de E-SEPG conferentie. Dit is het Europese SPI event, wat gehouden wordt van 11-14 juni in Amsterdam. SPIder donateurs en medewerkers van onze sponsors krijgen toegang op de toegangsprijs. Ook kunnen we, omdat we de officiële Nederlandse SPIN zijn, het lidmaatschap van de SEI met korting aanbieden. In deze koerier meer informatie hierover. Mocht je (bedrijfs)donateur willen worden, op de website staat hoe je je aan kunt melden. De minimale bijdrage verdien je zo terug. En, je laat zien dat je achter SPI en kwaliteit en SPIder staat.

Als laatste: SPIder is in zijn officiële 10e jaar als stichting. We willen dit vieren met onze leden. Hoe? Bijvoorbeeld op de conferentie, en in de plenaire sessies. Maar er is vast nog meer wat we kunnen doen. Mocht je ideeën hebt, dan hoor ik die graag!

Namens het bestuur,
Ben Linders
Voorzitter SPIder

## ■ Making Software Process Improvement Happen

*Anna Börjesson, Ericsson*

### Introduction

Software Process Improvement (SPI) has become one of the most widely used approaches to increase the capability of software organizations since its introduction by Watts Humphrey in the late 1980’s. Most software organizations do, however, struggle to find a good balance between SPI investments and returns. Far too many efforts fail or the resulting process is not used as intended or not used at all. The reasons are many, such as commitment and management attention, change management and reactions to change, knowledge barriers, and low deployment focus. This was also the situation within one development unit in the telecom company Ericsson in Gothenburg, Sweden.

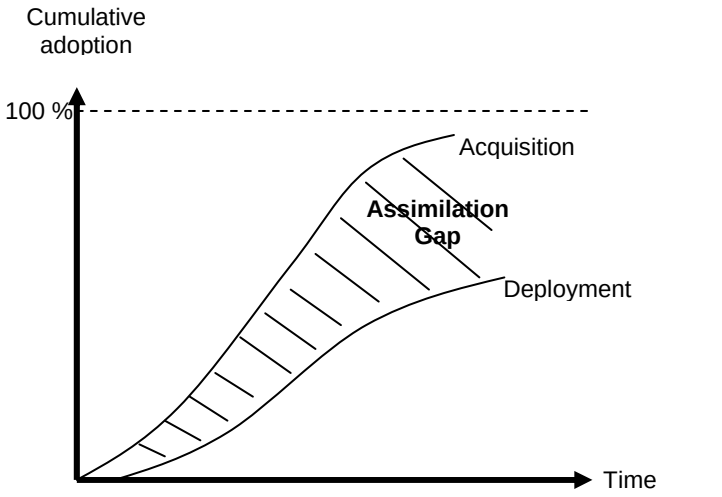



*Figure 1. The Assimilation Gap (Fichman and Kemerer, 1999)*

A first and necessary step towards SPI success is SPI implementation. To get satisfying return on SPI investments, new or modified processes must be implemented and used in practice. The benefits of new or updated processes that are not implemented in practice remain limited. There is unfortunately a gap between software engineering as it is actually performed and as it is intended to be performed as a result of SPI initiatives. While this gap is emphasized in current SPI literature and provides both understanding and approaches for SPI challenges, the awareness and knowledge about the assimilation gap (Fichman and Kemerer, 1999) and its practical impact on SPI is not well understood (See Figure 1).

Far too many SPI initiatives with well-reasoned ideas fail and the gap presented in Figure 1 between acquisition and deployment of technologies is problematic for software organizations. Organizations acquire new technology and run SPI initiatives to improve software engineering practices. When they fail to deploy the new technology, they also fail to get satisfactory return on investment of both the acquired technology and the SPI initiative. Software organizations, not aware of the assimilation gap, might develop an illusory picture of reality and they might miss opportunities to assure implementation and use of new technologies, i.e. they fail to make SPI happen. This research aims to decrease this gap through answering the main research question:

> RQ: How can SPI change agents improve implementation of software processes?

This question is approached from two different views. RQ1 focuses on improving SPI change agents’ understanding, while RQ2 focuses on improving the approaches that SPI change agents use.

> RQ1: How can SPI change agents improve their understanding of software process implementation?
>
> RQ2: How can SPI change agents improve their approaches to software process implementation?

These research questions are closely related, but their focus is different. Figure 2 describes how SPI change agent understanding and approaches relate to software process implementation, software process improvement and software development. Each arrow is double pointed to visualize the interdependencies and interactions between the elements. RQ1 focuses on SPI change agents and their understanding of software process implementation. RQ2 focuses on SPI approaches that SPI change agents use and how these affect software process implementation.

### Theoretical Background

Fichman and Kemerer argue for innovations in information technology to have a positive impact on quality and productivity, they must actually be

deployed. Yet, innovation researchers have known for some time that a new technology may be introduced amid great enthusiasm and enjoy widespread initial acquisition, but nevertheless still fail to be thoroughly deployed among many acquiring firms. Fichman and Kemerer use the terms acquisition and deployment to explain why new technology, e.g. new software processes, can fail during implementation. The difference between acquired and deployed technology becomes the assimilation gap (see Figure 1). Innovations may be widely acquired, but only sparsely deployed among acquiring firms. Fichman and Kemerer consequently argue for focusing on deployment of an innovation, to prevent an illusory perception of the diffusion process. Acquiring organizations may falsely believe they will receive net benefits just because they have acquired new technology. There is a considerable risk, they argue, firms do not understand and appreciate the difference between acquiring and deploying innovations. Knowing and accepting a substantial assimilation gap exists between acquiring and deploying, regardless of the precise reason why, can generate an important motivation for improvement in its own right.

Commitment plays an important role for the outcome of SPI initiatives (Humphrey, 1989). Management commitment to a change initiative tends to have great impact on deployment, especially compared to acquisition. Acquiring does not necessarily result in change, while deployment by definition means change, as successful deployment requires people to follow new processes and use new tools. Successful SPI depends on the commitment to the project from both managers and software developers.

Inappropriate change management is another factor explaining SPI failures. Quoting Weinberg (1997, p.48): ”A great deal of the trouble we have in experiencing change comes from a false set of expectations engendered by oversimplified change models. A more accurate change model helps by making our expectations more realistic”. Weinberg further explains this by giving the example about how organizations are often expected to do business as usual while a change is taking place.

Humphrey stresses enthusiastic, technically and politically capable, and dedicated resources with management support are necessary to reach successful SPI. Humphrey calls these dedicated resources change agents. Weinberg calls them change artists. He argues: “Change artists are the lubricant that makes change work”. Change artistry consists of knowing how to facilitate change, knowing what to change, knowing when and where to change, and knowing who should take what role to manage change. Weinberg claims to develop change artistry, theoretical learning, experimental training, and practice and experience are necessary. McFeeley stresses the importance of committing dedicated resources to drive the SPI work. McFeeley states a software engineering process group (SEPG), chartered by a management steering group, is necessary to facilitate the improvement activity. The SEPG candidates must be willing to serve as agents of change to the rest of the organization. McFeeley claims the agents of change must have support from senior management and be respected members of the organization.

Current SPI literature also provides us with a deeper understanding of the SPI change agent role. Kautz et al. (2001) contribute by discussing SPI from a change agent’s perspective (called process agents by Kautz et al.). They suggest an SPI change agent can work in combinations of four different perspectives (technical expert, facilitating participant, political agent, and individual therapist). The roles do not preclude each other.

The diffusion of innovation literature also recognizes the importance of the change agent role to accomplish successful diffusion. Rogers (2003) claims change agents’ success in securing the adoption of innovations by clients is positively related to the change agent’s level of contact with clients and the change agent’s credibility in the clients’ eyes. Fowler and Levine strengthen these claims through identifying SPI change agent roles and tactics as one of five key factors for successful diffusion of an innovation. The SPI change agent role is considered by most SPI and SPI related literature as necessary to accomplish successful change, which again is an essential prerequisite for SPI success. Rogers defines a number of central elements in diffusion: an innovation, communication through certain channels over time, the diffusion and the change it leads to in the target group, the uncertainty related to the many alternative ways in which the idea may be employed, the many ways in which communication may take place, and the many ways in which changes may occur. SPI deals in many ways with the same challenges.

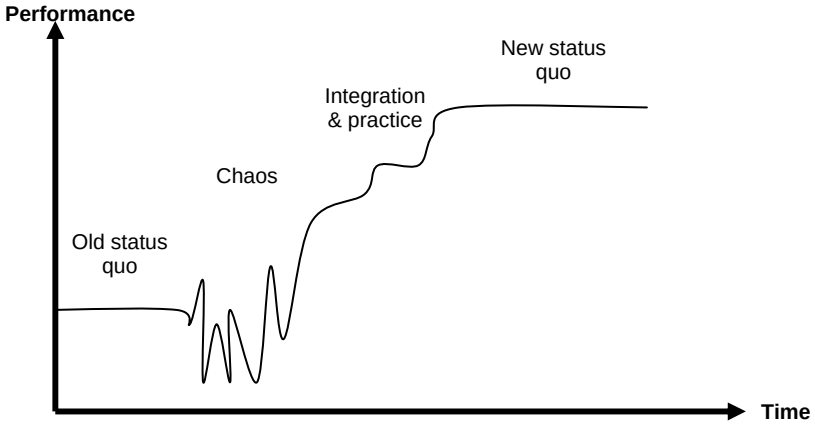



*Figure 3. The Satir change model (Weinberg,*

A complementary understanding of the fundamental challenge of SPI is expressed in models of organizational change. Weinberg’s Satir change model provides a deeper understanding of the change process (see Figure 3). The model presents four different phases that a change initiative passes through to reach higher performance. The first phase “Old status quo“ is distinguished by no change in performance. The second phase “Chaos“ is distinguished by performance going irregularly up and down. In the third phase, “Integration and practice“, there is a take-off towards higher performance and the fourth and last phase is

distinguished by higher performance, i.e. “New status quo”.

## Research Approach

This research project was initiated from an overall desire within Ericsson to benefit from having a close cooperation between industry and academia and from a specific desire to better understand how software processes could be implemented and used more effectively within Ericsson’s software engineering environment. Collaborative Practice Research (CPR) is a research approach that aims at fulfilling the dual objectives of both improving practice and contributing to the body of knowledge within research. The desire of having a close cooperation between industry and academia is through CPR partly accomplished by having an “insider” and “outsider” working in the same research project. CPR is a pluralistic methodology and makes use of three research approaches: practice studies, action research and experiments (see Figure 4).

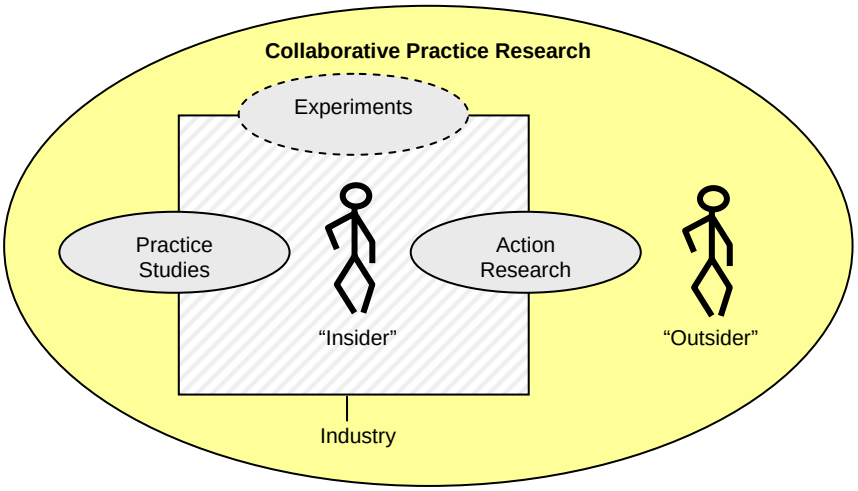



*Figure 4. CPR and its including approaches and roles (adapted from Mathiassen, 2002)*

CPR seeks to strike a good balance between rigor and relevance of the process and its findings by creating a fruitful collaboration between practitioners and researchers. Insiders (practitioners) and outsiders (researchers) working in close collaboration in a research team can take advantage of bringing their specific knowledge of identification, analysis, and interpretation together. The collaborative process may take different forms depending on how the collaboration between the insider and outsider is executed. A main concern, independent of the collaboration form, is, however, to establish a well functioning relationship between researchers and practitioners. There is a need for mutual respect of the different challenges that these two roles experience. The insiders can provide the research team with action experience and access to data and the outsiders can provide the research team with reflective and analytical capabilities. However, this kind of collaborative effort requires qualified and willing participants and a good match of knowledge interests.

| Contribution to Research |
| --- |
| 1. Software organizations can improve SPI change agents’ knowledge through action learning efforts. Such efforts tend to increase implementation success. |

## Discussion

RQ1: How can SPI change agents improve their understanding of software process implementation?

In January 2003, the SPI unit started an action learning effort including literature studies, workshop participation, SPI project meetings, and one-on-one coaching. This effort aimed to increase the SPI success rate by improving the change agents’ understanding and reflection capabilities in action. The change agents studied well-referenced SPI and SPI-related literature. This included practical SPI approaches, such as the workshop model and theories for understanding SPI, such as the Satir change model and the assimilation gap. Studying and reflecting on such sources should help explain, understand and improve the SPI change agents’ daily practice. The learning effort was organized according to the elements of action learning as defined by Marquardt. The action learning approach was found particularly helpful to find a way to combine learning and SPI change agents’ daily actions.

One of Humphrey’s six principles for successful software process change states: “Ultimately, everyone must be involved”. This principle calls for dedicated resources to drive SPI and software engineer participation in SPI. Diffusions can be driven by a demand pull where the reason for adopting a new technology is organizational needs triggered by a performance gap, or by a technology push where the reason for adoption is based on the espoused benefits of the new technology. In the specific case of process implementation we can distinguish between practice pull where the competence, commitment, and participation of the practicing software engineers is the key driver, and process push, where the diffusion is driven by the competence, commitments, and participation of the SPI change agents. When analysing and contrasting different SPI initiatives from a practice pull and process push view, we learned that SPI initiatives including considerable elements of practice pull and process push tend to be more successful than those that do not. The two attributes practice pull and process push, have provided an opportunity to broaden our understanding about participation in SPI initiatives in relation to SPI outcome.

Paulk argues the use of structured tactics to cope with change increases the likelihood of success. One approach we have found particularly helpful to understand SPI is Weinberg's Satir change model. The model consists of four phases (see Figure 3). The second phase is called "chaos" and this phase is distinguished by performance going irregularly up and down. People enter the chaos phase when getting exposed to change. Several activities aiming to achieve a new status quo can easily fall back into old status quo during the chaos phase. Many explanations have been given to understand the reasons for chaos like disruption caused by the change, neglecting old competencies as a result of the change, or that the change creates substantial fear by the would-be users of the new technology. However, people never entering the chaos phase will most likely not change. We have learned that SPI change agents can benefit from interacting more with the software engineers who are supposed to change their way of working. Their frustrations will lead to criticism and this criticism will become valuable feedback for the change agents and increase the likelihood of successful change. Even though the chaos phase can be considered rough by both SPI change agents and software engineers, we have found it beneficial in terms of getting valuable feedback on how to improve the new process. Hence we have learned it is important to recognize the chaos phase as an indicator of software process implementation progress instead of viewing chaos as something that is hard to manage. This understanding can help SPI change agents focus on implementation and stay current in SPI initiatives when the going gets rough.

| **Contribution to Research** |
| --- |
| 3. SPI initiatives exposed to criticism and feedback are likely to experience chaos. This chaos should be recognized as an important opportunity for improvement. SPI initiatives experiencing chaos tend to have higher process implementation success in the end than those that proceed unnoticed. |

Software organizations constantly need to react to market dynamics, new customer requirements, technological innovations, and mergers between software companies. To be successful, SPI initiatives must therefore be organized, managed, and executed in ways that allow them to effectively sense-and-respond to both expected and unexpected events in their environment. Software researchers and practitioners have over the past years adopted agility principles and approaches to respond more effectively to dynamics in customer demands. Dove defines agility with two key elements: response ability and knowledge management. Dove argues an organization has to manage knowledge effectively to thrive in a continuously changing business environment. We have learned the response ability of SPI initiatives is fragile or at best opportunistic when no mechanisms are in place to help identify, assess, and cope with environmental dynamics. A fragile organization has both low proactive proficiency and low reactive proficiency, while an opportunistic organization has low proactive proficiency, but higher reactive proficiency. The RequisitePro case provides an understanding for why and how agile capabilities can help SPI initiatives progress faster. This understanding has helped us broaden our view on the relation between SPI and parallel organizational dynamics. We argue that this particular SPI initiative would have benefited from organizational mechanisms that encourage sensing, coordinating and responding to the organizational mergers that took place. One such set of mechanisms is proposed by Haeckel, including the sense-and-respond cycle.

RQ2: How can SPI change agents improve their approaches to software process implementation?

To drive SPI work towards success, organizations must commit and manage their SPI change agents accordingly and they must constantly motivate software engineers to participate and contribute. Any significant change can create substantial fear and uncertainty throughout the organization. It is therefore essential that SPI change agents possess change management skills and work closely with software engineers to solve potential difficulties that arise, regarding both the new process and the change process itself. This kind of collaboration requires, however, that the SPI change agents spend considerable effort on deployment activities. Fichman and Kemerer claim ignoring deployment activities can create an illusory picture of SPI implementation success. This knowledge has been particularly important to understand the need of focusing on deployment in SPI. One of Humphrey's six basic principles for successful process change states: "SPI requires investment" and one such investment is SPI change agents spending effort on deployment activities. From our research at Ericsson we have learned SPI initiatives spending more effort in the later phases of the IDEAL model, that focus on deployment (i.e. the pilot, refine and implement solution steps in the IDEAL model), tend to have higher software process implementation success than those that do not. When SPI change agents spend effort on deployment activities, they get the opportunity to collaborate closely with the software engineers and they become deeply engaged in their problems. This gives the SPI change agents the possibility to understand and improve the solution as the SPI initiative progresses.

| **Contribution to Research** |
| --- |
| 4. SPI initiatives lacking agile capabilities cannot effectively sense and respond to new employees as organizational mergers take place. Lack of agile capabilities tends to affect implementation success negatively. |

To achieve software process implementation success, it is vital that SPI change agents have a positive and collaborative relationship with the software engineers. Positive customer relations will help the SPI change agents better understand engineering practices and problems, and thus develop more useful processes. A good relationship also helps software engineers overcome resistance

to change. When the old status quo is challenged, they often become defensive and alienated. Rogers argues the same through stating change agents' success in securing the adoption of innovations is positively related to the change agent's credibility in the clients' eyes. We have learned that set-ups, where SPI change agents can develop a close relationship with their customers, tend to have higher software process implementation success than those that do not. In particular, dedicated SPI initiatives that focus on supporting one software engineering project tend to be more successful than generic SPI initiatives that focus on supporting several software engineering projects. Generic approaches make it difficult for SPI change agents to build and maintain good customer relations — there are simply too many different relationships, needs, and requirements. In dedicated approaches, SPI change agents and software engineers can work closely together and focus on the project's specific challenges.

| Contribution to Research |
| --- |
| 6. Dedicated SPI initiatives supporting one engineering project tend to have higher implementation success than generic SPI initiatives supporting several engineering projects. |

Successfully changing software practices requires learning, and helping organizations learn is by no means easy. Iterations support learning by allowing SPI change agents to correct failures and modify processes based on practical experience. Although Humphrey does not use the word iteration, he does discuss the importance of performing SPI work in steps and repeatable sequences. Several SPI approaches emphasize iterative development, including the IDEAL model and methods that implement Plan-Do-Act-Check cycles. We have learned that SPI initiatives executing several iterations tend to have higher software process implementation success than those only executing one. Executing several iterations have other positive impacts as well. Most SPI initiatives must pass the chaos phase to enter the integration and practice phase to implement a new or modified process successfully. SPI initiatives only executing one single iteration will therefore seldom enter the phase of using the new process as an integral part of engineering practices. Thus, the process will not be adopted as intended. SPI initiatives only executing one single iteration are likely to leave the software engineers in a situation, where they have been exposed to the change and experience many types of negative reactions to the change, but where there are no SPI change agents available to discuss potential improvements and to help the software engineers overcome resistance to the change. We have found Weinberg's Satir change model (see Figure 3) particularly helpful to learn more about the relation between SPI implementation success and managing reactions to change.

A failure to diffuse an innovation can stem from not recognizing and understanding the chasm in the technology adoption curve. There is a gap between acquired and deployed technologies in SPI and Moore's chasm partly explains this gap. Innovators

| Contribution to Research |
| --- |
| 7. SPI initiatives executing several iterations, in which practitioners are exposed to change in ways that make them understand how the new process will impact their daily work, tend to have higher implementation success than SPI initiatives only executing one iteration. |

and early adopters are prepared to accept bugs and glitches of a new technology. The early majority wants, however, technology to enhance, not overthrow, the established ways of working. Because of these incompatibilities, the early adopters do not make a good reference for the early majority. We have therefore developed a new SPI tactic, the guerrilla tactic, which is based on Moore's "down the bowling alley" tactic, to address these incompatibilities. This tactic focuses on establishing a trustworthy reference group and working closely with both the group as a whole and its individual members. The guerrilla tactic is an expression of the core values of the Agile Manifesto and the tactic can be viewed as an agile improvement practice useful when improving software organizations in an ever-changing business environment. The guerrilla tactic facilitates sense-and-respond activities through the close cooperation between the SPI change agents and the practitioners (i.e. reference group). We have learned that the agile guerrilla tactic can help SPI change agents cross the chasm and successfully implement new or changed processes.

To be successful, software organizations must be organized, managed, and executed in ways that allow them to sense-and-respond effectively to

| Contribution to Research |
| --- |
| 8. SPI change agents can use an agile guerrilla tactic to cross the chasm by successfully connecting to early majority adopters. SPI change agents using the guerrilla tactic tend to be successful in implementation of new or modified processes. |

unpredictable events in their environment. To understand more about SPI agility, we have identified and compared two different SPI project tactics and studied their use and outcome in 18 SPI initiatives at Ericsson. One tactic is generic - aiming for the development of organization-level software process elements. We call this the supertanker tactic. The other tactic is dedicated - aiming for unique solutions for particular development projects. We call this the motorboat tactic. These two tactics have helped us understand about the challenges involved in the set-up of SPI initiatives in relation to SPI outcome in dynamic business environment.

The CMM mindset is rooted in the CMM and the current CMM Integration (CMMI). These models assume good processes are predictable and lead to good products. The CMM mindset is based on an ambition to create sustainable progress by bringing processes under statistical control. This emphasis on control can, however, overshadow or conflict with a need to be flexible to respond to change effectively. The increased speed of technological and market changes have generally led to considerable interest in how organizations can effectively respond to changing environments. Haeckel's sense-and-respond cycle offers a specific

approach to implement learning processes in dynamic environments. Such models are not obvious in the CMM mindset. McFeeley's cyclical IDEAL model does however provide opportunities for sensing and responding during the diagnosing and learning stages. This model assumes in this way that learning has to take place regularly. In contrast, the CMM model does not explicitly emphasize learning by establishing a dynamic business environment with constant sense-and-respond capability. We have learned SPI projects based on a traditional CMM mindset tend to have low SPI implementation success in dynamic business environments.

| Contribution to Research |
| --- |
| 9. The traditional CMM mindset assumes a rather stable environment and has become a supertanker in today's dynamic business environment. SPI initiatives practicing the CMM mindset tend to have low SPI implementation success in dynamic contexts. |

The supertanker tactic is centralist and top-down with an ambition to build and enforce organization-level standard processes. Process tailoring is the response to particular project conditions, and there is strict process enforcement and separation of thinkers from doers. The SPI governance structure emphasizes, in this case, overall coordination of projects and organization-wide institutionalization of standard processes. Learning primarily takes place on the overall SPI level through continuous sense-and-respond cycles that identify current weaknesses, initiate new efforts, and implement their results organization-wide. The supertanker tactic is built on the CMM mindset. We have identified an alternative motorboat tactic that is decentralist and bottom-up with an emphasis on project and team level standardization of processes. It focuses on emergent processes that promote discretion and latitude at the level of the engineer. It also emphasizes practice pull to cultivate processes in response to situational opportunities, and it relies on mentoring and coaching software engineers by embedding SPI change agents directly into software projects. Key to the motorboat tactic is the support of adaptive SPI practices. The governance structure emphasizes coordination between SPI change agents and software engineers allowing for local authority in each initiative. Learning primarily takes place within initiatives through continuous sense-and-respond cycles that identify current weaknesses, initiate new efforts, and implement these as the project evolves and delivers its results. The motorboat tactic is a critique to the CMM mindset assuming one-size-does-not-fit-all. The motorboat tactic is built on a belief that deployment focus, dedicated initiatives, and iterative SPI work are success factors for successful process implementation.

| Contribution to Research |
| --- |
| 10. SPI initiatives using a motorboat tactic based on an agile mindset tend to have higher SPI implementation success than SPI initiatives using the supertanker tactic. |

## References


Fichman, R. G. and Kemerer, C. F. (1999) The Illusory Diffusion of Innovation: An Examination of Assimilation Gaps. Information Systems Research, Vol. 10, issue 3, pp. 255-275.

Humphrey, W. S. (1989) Managing the Software Process, Reading, Massachusetts: Addison Wesley.

Kautz, K., Hansen, H.W. and Thaysen, K. (2001) Understanding and Changing Software Organisations: An Exploration of Four Perspectives on Software Process Improvement. In in Diffusing Software Product and Process Innovations, M. A. Ardis and B. L. Marcolin (Eds.), Boston: Kluwer Academic Publishers, pp. 87-109.

Mathiassen, L., Pries-Heje, J. and Ngwenyama, O. (2002) Improving Software Organizations – From Principles to Practice, Upper Saddle River, New Jersey: Addison-Wesley.

Rogers, E. M. (2003) Diffusion of Innovations, Fifth Edition, New York: Free Press.

Weinberg, G. M. (1997) Quality Software Management volume IV – Anticipating change. Dorset House Publishing, New York, USA.


*Note*

*This article is based on the doctoral dissertation from Anna Börjesson in 2006, and has been edited by Ben Linders to fit into the available space. For more information, feel free to contact the author at*

anna.borjesson@ericsson.com.
IT University of Göteborg & Ericsson AB


## ■ Data voor de 10e SPIder conferentie:

*Energyzing Improvements*

- Korting deelname: Tot 15 juni
- Conferentiedag: 2 oktober 2007

Zet deze datum in je agenda!

www.spiderconferentie.nl

Stuur uw ideeën voor een onvergetelijk feest t.a.v. het 10-jarig bestaan van SPIder naar de voorzitter: voorzitter@st-spider.nl

## ■ Infotentie

*Informatieve (bijna) advertenties*

**E-SEPG Conferentie, Juni 11-14, Amsterdam**


*The European SEPG® conference brings together world leaders, innovators and practitioners to explore the role and practice of process improvement in competitive global enterprise. It demonstrates how organisations are developing skills and approaches to meet the needs of rapid change, and how they are incorporating multiple models into their improvement efforts. Kick-start your improvement efforts; perfect your measurement techniques; master complex initiatives encompassing multiple domains and models; receive cutting-edge knowledge in software and systems development, services and acquisition process. At the European SEPG, learn from those who have found innovative answers to the challenges you face.*

SPIder is wederom co-chair van deze conferentie en aanwezig met een stand. SPIder donateurs krijgen een **korting van 50 euro** op de toegangsprijs, met een SPIder kortingscode (aan te vragen bij het secretariaat, info@st-spider.nl).

**SEI Membership, aanbieding t/m 30 juni 2007**

*The SEI Membership program was established in 1992 to provide opportunities for members to advance, network, and learn. SEI members are leaders in software engineering and related disciplines and include CEO-s, directors, and managers from both Fortune 500 companies and prominent government organizations. Many members have used their SEI membership to increase their professional standing and affiliations. The benefits and services of SEI membership include access to members-only opportunities, as well as discounts on SEI products and services.*

SPIder donateurs kunnen voor een gereduceerde prijs lid worden van het SEI: Een korting bij aanmelding van **$150,- in het eerste jaar en $50,- in de volgende jaren**. U betaalt dan $200,- in het eerste jaar en daarna $150,- per jaar.

Aanmelding via spin@sei.cmu.edu met vermelding van "Member Dutch SPIN (SPIder)".

## ■ SPIder conferentie

De 10e SPIder conferentie komt eraan. Thema dit jaar is

*Energyzing Improvements*:

Waar zitten de Machiavelli's onder de verbeteraars?"

We beloven u een dag vol inspirerende en interessante ervaringen op het gebied van de menselijke factor in verbetertrajecten. Ook zetten we u door middel van het thema aan het denken om tot creatieve prestaties te komen.

Een breed scala aan onderwerpen zal worden belicht, waaronder:
- opleiding van professionals, hoe doe je dat effectief?
- wat is dat eigenlijk: op een professionele manier verbeteren?
- welke krachtenvelden spelen rond verbetertrajecten?
- waarom is het aantrekkelijker een held te zijn dan een professional?

Kortom een inspirerende dag die naast het onderstaande programma volop kansen biedt om ervaringen uit te wisselen en te netwerken!

De presentaties komen vanuit de praktijk, in een vorm die ze toepasbaar maakt in uw eigen organisatie. Ze laten zien welke SPI aanpak gewerkt heeft en welke niet, de key succes factoren en de resultaten. Met de resultaten kan de effectiviteit en efficiency van SPI in uw bedrijf verbeterd worden. In één dag bent u weer up-to-date met SPI.

De conferentie vindt plaats op

dinsdag **2 oktober 2007** in
conferentiecentrum De Reehorst in Ede.

Meld u aan via www.spiderconferentie.nl.

Let op: Tot 15 juni a.s. geldt extra korting.

Carnegie Mellon | Software Engineering Institute

## Programma:

09.30-10.00 Ontvangst

10.00-10.15 **Opening door de dagvoorzitter**
*Ben Linders (voorzitter Stichting SPIder)*

10.15- 11.05 **Keynote: Professionaliteit: van software engineering naar menselijke waarden**
*Jeroen van den Hoven (TU Delft).*

11.05-11.20 **Pauze**

11.20-12.10 **Veranderen met zachtheid**
Rogier Guns (Philips)

12.10-13.30 **Lunch**

13.30-14.20

| **Coaching** ***Track A*** | **SPI** ***Track B*** | **Agile** ***Track C*** |
|---|---|---|
| **Hoe krijg ik een blije opdrachtgever?** Daniel van der Gaag (Sogeti) | **Soft skills binnen een Six Sigma project** Adriaan Muller (UNC Slim) | **Agile Development / retrospectives bij Ericsson Nicole Belilos** (Topic) & Veerle van den Bossche (Ericsson) |

14.20-14.25 **Wisselpauze**

14.25-15.15

| **Coaching** ***Track A*** | **SPI** ***Track B*** | **Agile** ***Track C*** |
|---|---|---|
| **Coaching; de brandstof voor je verbeterprogramma** Erik van der Vliet (LogicaCMG) | **Lessons learned from good process improvement attempts: it boils down to humans** Viktor Clerc (DNV-CIBIT) | **Meanderen of kanaliseren? Martin Mermans (Philips) &** Rene Krikhaar (Vrije Universiteit & ICT NoviQ) |

15.15-15.45 **Pauze**

15.45-16.35 **Cultuur als factor in communicatie en verandering**
*Nelke Galema (ITIM Result)*

16.35-17.50 **Keynote: Het E=MC$^2$ van veranderen**
*Remco Claassen (ADVANCE Leadership development BV).*

17.50-18.00 **Afsluiting van de conferentie door de dagvoorzitter**

18.00-20.00 **Diner buffet (optioneel)**

20.00 **Slot**

## ■ SPIder en LinkedIn

SPIder is sinds kort een groep in het netwerkprogramma LinkedIn.

Je kunt je aanmelden door de volgende link te volgen. De moderator van de groep zal de aanvraag al dan niet honoreren.

**Link to join:** SPIder group:
http://www.linkedin.com/e/gis/3131/61181E44E863

Als het gelukt is zie je bij de andere SPIder leden ook het logo verschijnen.

## ■ Evaluatie 6 december 2006 Q workshop

*Johan Zandhuis, SYSQA*

**Kwaliteit begint bij jezelf**! Dat stond in de uitnodiging voor een interactieve workshop die 6 december 2006 werd georganiseerd door de Q Society. Aan de hand van prikkelende stellingen werd gediscussieerd over de zin en onzin van 'kwaliteit'. Doordat de groep van deelnemers nu eens niet bestond uit puur kwaliteitsmedewerkers waren de discussies levendig en vooral basis voor nieuwe inzichten en beter wederzijds begrip tussen 'zij-die nog-altijd-maar-niet-snappen-hoe-belangrijk-kwaliteit-is' en 'zij-die-alleen-maar-oog- hebben-voor-kwaliteit' en iedereen daar tussenin. De algemene terugkoppeling was dan ook dat men door de workshop aan het denken was gezet en dat velen met minimaal één nieuw inzicht of goed idee huiswaarts ging.

In een eerdere uitgave van de SPIder Koerier is door Martin Muller al kort verslag gedaan van deze workshop. Het eindverslag is een consolidatie van de resultaten van de afzonderlijke sessies. Ondanks dat het al weer een poosje geleden is brengen we onderstaand een samenvatting van de stellingen en de reacties hierop.

De geponeerde stellingen luidden als volgt:

1. De introductie van 'Quality' werkt contraproductief!

2. Het kwaliteitsprobleem is onvoldoende zichtbaar.
3. De oorzaken van onvoldoende kwaliteit zijn niet bekend.
4. We leggen kwaliteitsvraagstukken niet aan de juiste personen voor.

Gaandeweg de workshop bleek dat na een aantal stellingen te hebben bediscussieerd bij een volgende stelling vaak dezelfde oorzaken, redenen en oplossingen naar voren kwamen.

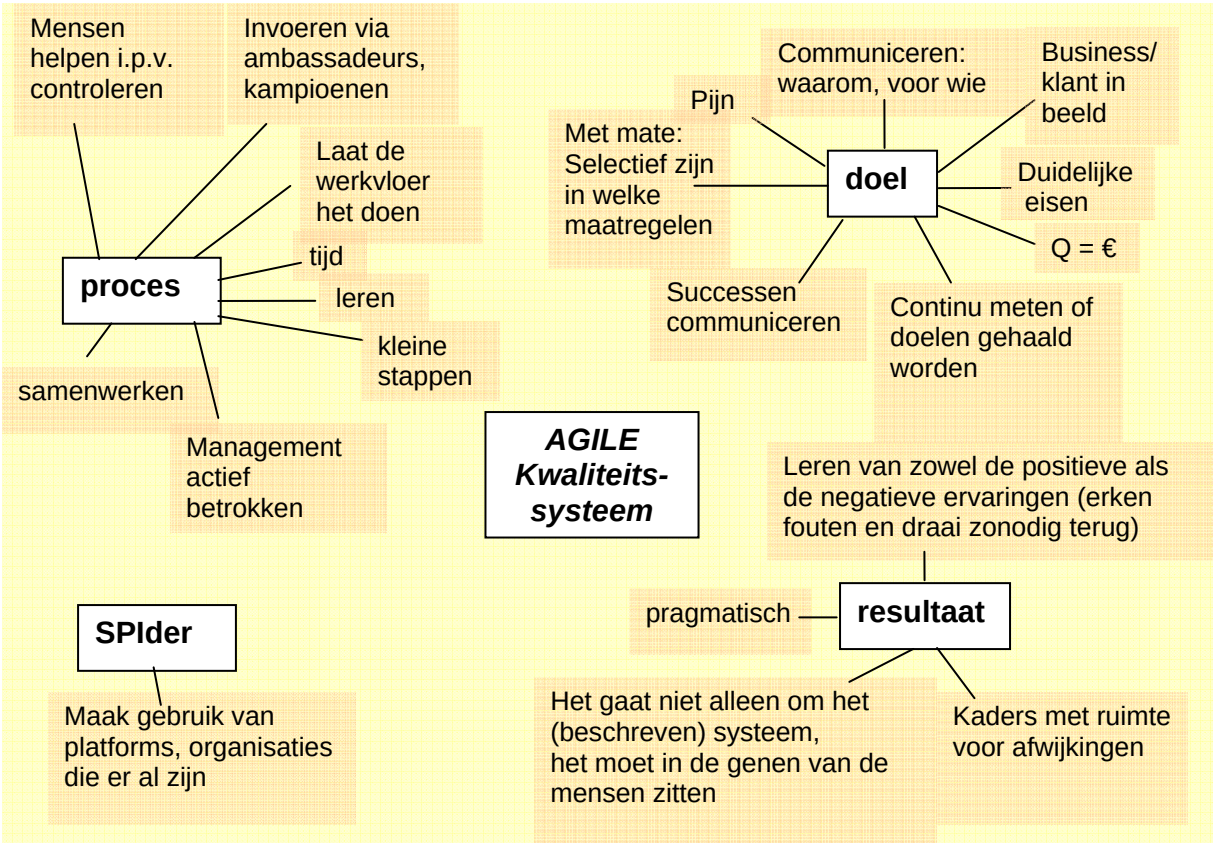
In de discussies werd duidelijk dat de verschillende deelnemers een verschillend begrip hadden van wat kwaliteit is. Kwaliteit is veelal opgevat als het invoeren en gebruiken van een kwaliteitssysteem. De relatie met productkwaliteit is regelmatig gelegd, waarbij het uiteindelijk gaat om de (kwaliteit van de) *business*producten. Er was overeenstemming over het feit dat kwaliteit heeft te maken met het bereiken van *business*doelen.

In het algemeen bleek er consensus te zijn dat kwaliteit introduceren productief is als je het maar selectief doet en de invoering geleidelijk laat verlopen. Tevens dient kwaliteit gemeten te worden om vast te stellen wat het nuttig effect van kwaliteitsmaatregelen is.

Hierbij moet vooral gekeken worden naar het (nuttig) gebruik in de praktijk van een kwaliteitssysteem en de definities van productkwaliteit.

Algemeen was men het er over eens dat introduceren van afspraken over kwaliteit initieel geld kost en pas later rendement oplevert. Anders gesteld: het is een investering. Daarentegen is de pijn (gevolgschade ten gevolge van fouten, rework, omzetverlies, klantverlies) soms onzichtbaar, soms wel zichtbaar maar wordt veelal ontkend. Als het al zichtbaar is valt het niet inzichtelijk en meetbaar te maken, of soms pas op langere termijn. Of zoals een van de deelnemers het verwoordde: *"Je voelt de pijn pas echt als je gezien hebt dat het ook goed kan gaan"*.

Maatregelen die genomen worden moeten in beperkte tijd doorgevoerd kunnen worden, moeten snel baten opleveren en moeten onderdeel zijn van de visie die de organisatie op kwaliteit heeft. Het ontwikkelen en onderhouden van een kwaliteitsysteem moet onderdeel zijn van een cyclus: het is geen eenmalige activiteit.



Kwaliteitssystemen komen in de praktijk vaak als te mechanisch over. Enkele aardige uitspraken in dat kader:

- *"kwaliteitsprogramma's zijn een excuus om niet echt te veranderen"*
- *"Ik heb liever een bedrijf dat niet gecertificeerd is als leverancier, die hebben meer aandacht voor me."*

Daar waar medewerkers en managers kwaliteitsafspraken niet meer ervaren als hulpmiddel om de juiste dingen juist te doen en waar de klant niet meer in beeld is, daar blijkt het averechts te werken. Methoden en technieken die niet worden begrepen leiden tot een verkeerde uitleg, verkeerde aanpak en uiteindelijk tot een kwaliteitssysteem dat niet meer begrepen en gedragen wordt. Mensen die zich betrokken voelen bij de kwaliteitsafspraken leven deze na en leveren bijdrage tot verbetering.

Tenslotte moet het proces voor ontwikkeling en onderhoud van een kwaliteitssysteem en het systeem zelf ***agile*** zijn. Hiermee wordt bedoeld dat de doelen van een organisatie veranderen en dat het kwaliteitssysteem zich continu moet aanpassen aan de doelen van de organisatie en dus geen belemmering mag vormen voor het wijzigen van de organisatiedoelen.

Schematisch zijn de conclusies op de stellingen hiervoor samengevat:

Het totaal overzicht van de uitkomsten van de workshops is te vinden op de site van de SPIder (st-spider.nl).

Bijdrage van Johan Zandhuis, SYSQA

## Ten Misconceptions about Product Software Release Management explained using Cost/Value Functions

*Slinger Jansen, Sjaak Brinkkemper*
*Information Sciences Institute Utrecht University*

### Abstract

*The decision for a young product software vendor to release a version of their product is dependent on different factors, such as development decisions (It feels right), sales decisions (it sells well now), and quality decisions (the product is stable). Customers of these products, however, are much more cost oriented when deciding whether to update their product or not, and will look mainly at the cost and value of an update. Product software vendors would gain tremendously if their release package planning method was supported by a similar cost/value overview. This paper presents cost/value functions for product software vendors to support their release package planning method. These cost/value functions are supported by ten misconceptions encountered in seven case studies of product software vendors that these vendors had to adjust during their lifetime.*
*Finally, a number of cost saving opportunities are presented to enable quicker adoption of a release and thus shorten release times and customer feedback cycles*

.

### 1 Introduction

Product software release planning has been characterised as a "wicked" [4] and complex [1] problem for which no perfect solution exists. One part of release planning, release package planning, is often underestimated due to its seemingly innocent and uncomplex nature. Product software vendors that do not have much experience in release planning often publish their releases because a team of experts within the organization deems the release good-enough, which results into some releases that are hardly adopted by customers, whereas others are much more popular.

Simultaneously, releases are created often during the lifecycle of a product, which suggests that processes such as release creation, release publication, informing the customer of a new release, and updating are repetitious processes that must be automated as much as possible, to ease both customer update effort and vendor release creation effort. Decreasing this effort results into customers that are more willing to update, and vendors who are more willing to release regularly, in turn improving quality, as suggested by the agile development methods, such as extreme programming [2]. However, from a number of case studies performed in the past it is found that product software vendors generally do not sufficiently plan their releases [9].

Software product release management is the storage, publication, identification, and packaging of the elements of a product. Release package planning, which is part of the release planning process, is the process of defining what bug fixes and features are included in a release package and the process of identifying these packages as bug fix, minor, or major updates, taking into account releases that have been published in the past and the possible update process required to go from one release of the product to another release. First, an update is a package that promotes a customers release to a newer release. Secondly, a bug fix update contains only bug fixes, a feature update contains only new features, and minor and major updates contain both bug fixes and new features. The distinction between minor and major updates is usually that major updates change structural parts of a product, such as the

architecture or the data model. Our view of software evolution described here is similar to Rajlich and Bennet's staged model [3], which addresses evolutionary changes (minor and major updates) first, and then continues to see patches (bug fixes) until a release is phased out and closed down.

This paper highlights release package planning by specifically focussing on the cost and value of creating a new release for a software vendor, and the cost and value of an update to a customer. Two cost/value functions are presented that enable a product software vendor to estimate whether a release will actually be downloaded and installed by its customers. Also, two cost/value functions are presented to help a software vendor decide whether the next release will be marked a bug fix, minor, or major release.

The presented cost/value functions provide an extra check before publishing a release for product software vendors. In that, the presented decision method is a useful extension to product roadmapping methods, such as the one presented for small product software businesses [15], the method that supports the product software knowledge infrastructure [16], and other methods that support release planning [14]. Section 2 presents and describes the cost/value functions. Section 3 describes ten misconceptions encountered in seven case studies that support the cost/value functions as a valid release package planning method. In section 4 method are described to save either customer update costs or vendor side release costs and in sections 5 and 6 we discuss the presented method and draw our conclusions.

## 2 Defining the cost/value functions

This section describes the cost/value function for both the customer when updating its software and the vendor when releasing a new version. These functions separately describe the cost of an update for a customer, the value of an update for a customer, the value of a new release for a vendor, and the cost to create the release for the vendor. The functions are based on case studies performed at seven organisations [9]. Also, the customer functions are based on different papers from Enterprise Resource Planning (ERP) application updates and migrations [13], and a recent case study we performed at a Dutch content management systems vendor that also does updates and migrations for customers [6].

### 2.1 Customer Functions

A customer will base its decision to update a software product on a number of factors. First and foremost, the customer is interested in the value the update represents for her. This value can be of many different forms, such as the addition of a new level to a game providing the customer with more entertainment, a complete new production planning module to an ERP package saving the customer many millions, or a bug fix to an open source operating system that fixes a security leak in a web server. Simultaneously the customer will take the cost of updating into account. Such cost can be the downright effort of downloading and installing the new level for the game, the downtime of the ERP system during the update costing the company many millions, or the customisation to the web server that cannot function next to the security update to the operating system.

A customer's value of an update is defined as function 1 in figure 1.

The function defines the value of an update to a customer as the value of new features the customer will use plus the value of the removal of previous workarounds. The new features include those features that have been added to the new release, but also those that simply did not work and for which a workaround was not available. Before a customer decides to update, however, the customer will calculate the cost of an update to see if it is really worth it. The cost function is defined as function 2 in figure 1.

This function defines the cost of an update as the cost of downtime of a product, the cost of training for the people using the new/changed functionality, the cost of effort put into the update process, the cost of functionality that was removed from the release or the cost of customisations that can no longer be used after the update, and finally the cost of the payments to the vendor for the update. For a customer to make the update decision, *Cval* must exceed *Ccost* (see condition 3 in figure 1), especially when taking into account that the resources that are required to perform the update normally perform other value adding tasks. It is quite surprising to see that many vendors do not invest constantly into reducing these costs for the customer, especially since in most of our case studies up to 70 percent of revenue was coming from existing service contracts and only 30 percent from new customers. Also, when a vendor sees that *Ccost* exceeds *Cval* for a large number of customers, releasing an update becomes essentially useless, unless the vendor hopes to attract a large number of new customers. This seems improbable though, since if current customers are not interested in the product, why would new ones be?

$$Cval(update) = value(newFeatures) + value(removalOfWorkarounds) \quad (1)$$

$$Ccost(update) = \begin{cases} cost(downtime) + cost(training) + cost(effort) + \\ cost(lostFunctionality) + cost(paymentToVendor) \end{cases} \quad (2)$$

$$Cval(update) > Ccost(update) \quad (3)$$

**Figure 1. Customer Cost/Value Functions**

### 2.2 Vendor Functions

For a vendor the value of an update is much harder to calculate, especially because it involves estimating how many new customers are attracted with the new release and how many customers are actually prepared to update. A vendor's value of a new release are new customers attracted by the release that specifically targets a

new market, the reduction in support calls due to a bug fix to a commercial operating system, or a customer that pays the vendor for an update of their ERP product. The function for a vendor's value of a new release is defined in 4 in figure 2.

$$Vval(newrelease) = newCustomers * priceNewRelease + oldCustomers * priceOfUpdate + costReduction(support) \quad (4)$$

$$Vcost(newrelease) = \begin{cases} cost(development) & + & cost(updatingCurrentCustomers) & + \\ cost(increasedSupport) & + & cost(marketing) & + \\ cost(deliveryToCustomers) & + & cost(releaseCreation) & \end{cases} \quad (5)$$

$$Vval(update) > Vcost(update) \quad (6)$$

**Figure 2. Vendor Cost/Value Functions**

The resulting function describes the value of a new release as the amount of new customers times the price of a new release, the cost reduction in support calls due to the fixes in the new release, and finally the current customers who are prepared to update against reduced cost. Calculating the *Vval* is hardest, mostly because it involves estimating the amount of new customers and estimating how many customers are willing to update from any previous version, which might introduce different prices for the different updates. If the release contains a large number of bug fixes there might be a cost reduction in support costs. However, if many new features have been introduced, this reduction might be cancelled out by the cost increase in support. This brings us to the *Vcost* function for a new release in 5 in figure 2.

The *Vcost* function is defined as the cost of development of the functionality and bug fixes for the new release, the cost of updating the current customers, the cost of increased support questions relating to the new release, the cost of marketing, the cost of delivering the new release to customers, and finally the cost of creating the release. The cost of updating current customers includes such things as update tools [10], renewing their licenses, and possible support questions that arise during the update process. The cost of marketing includes informing current customers of the new release, the printing of folders, paying sales personnel to travel around the world, creating release notes, and maintaining the product's website. The cost of delivering the update to customers encompasses the creation of the delivery medium (CD, DVD, floppy, USB-stick, website, etc.), the assembling of all artefacts, the possible translations of the products language files, and the completeness checking of the release.

The functions shown in this section tend to change largely when looking at either a bug fix, a minor, or a major release. In the case of a bug fix, contrary to a major release, no new boxes need to be created by a vendor. The decision for either a bug fix, minor, or major release can be made using these functions. If the reduction in support costs justifies the effort put into fixing a number of bugs, a bug fix release is justified. If the reduction in support costs does not justify the effort put into fixing a number of bugs and the addition of functionality, you might want to earn it back by making the next release a minor release. If the vendor feels that the next release should generate more revenue from new customers and old customers as well, this might be a justifiable case for a major release. Of course this is not a hard science. Especially in the case a bug cost a disproportionate amount of time to fix, it might not be justifiable to publish a minor release. A question the vendor must ask itself then is whether it was worth it to try and fix the bug in the first place.

## 3 Ten Misconceptions about Product Software Releasing

Here we present ten misconceptions seven product software vendors had somewhere in their lifetime that they had to change. These misconceptions are generally strategic misconceptions that beginning software vendors can easily have about product software release management. The value/cost formulas presented in this paper support the lessons learnt presented here.

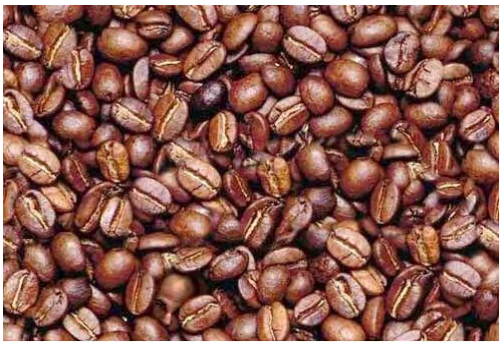
**1. Customers want to stay up-to-date** - It is important to realize that a customer of a software product uses it only to make life better. If a newer release does not provide the customer with new functions, why would she update? When, for instance, was the last time you updated the software on your microwave? Or in your car? To quote one of the case study participant's customers "Their software supports our business process perfectly. Some of the workarounds are strange, but as long as we don't have to invest in the ghastly process of updating, we're happy." This is a clear example of where *Ccost* exceeds *Cval*.

**2. Customers must stay up-to-date** - to guarantee success of a product software vendor. The misconception is demonstrated by the example of a Dutch content management system manufacturer, where customers use versions from years back who never updated due to the large amounts of customisations and complex update process. These customers, however, don't feel limited in their use of the product however, and will update when they require new functionality. Once again *Ccost* exceeds *Cval*.

**3. Release *n + 1* is better for a customer than release *n*** - Many of a Dutch bookkeeping software vendor's customers were still using the DOS based version of their product until 2005 when the vendor declared it would no longer support the DOS version. When attempting to update all these small entrepreneurs to a GUI based version the main complaint was that the graphic interface was less intuitive than their previous DOS versions. The bookkeeping software vendor ended up implementing the same keystroke combinations that were typical of the DOS era, into their GUI based client. Even though from the vendor's point of view their update to the GUI based version was necessary, customers could have worked with the DOS version for at least the next ten years and considered *Ccost* to be larger than *Cval*.

**4. Fixes can be postponed to the next major release** - A typical mistake to make is to postpone bug fixes for later releases, hoping to save the effort of having to implement the fix into multiple releases. This works fine if customers are eager to update, and the next major release is around the corner. However, in one of the case studies performed in 2004 we encountered a vendor who postponed many bug fixes to its next major release. The major release, planned for early 2005, still has not been released today. Many of the bug fixes had to be backported to keep customers satisfied. This is a clear example where *Vcost* seemed to be lower than *Vval*, but actually was not.

**5. Workarounds are evil** - Once again, as long as *Ccost* exceeds *Cval*, workarounds are a nice solution to a problem that would otherwise require a large investment from an organisation or person. An example of this is the Internet Explorer workarounds for stylesheets. Quite often stylesheets will look different on Internet Explorer 6 than other browsers, due to a bug. It is common knowledge, however, that Internet Explorer's interpreter can be fooled by adding specific characters to the code of a stylesheet. Microsoft has chosen not to fix this bug until Internet Explorer 7, mainly due to the fact that everyone is aware of the workaround and too many customers would need to be updated.



**6. Customers always want new features** - This common misconception is that any release can contain new features, since the customer should be happy with (possibly) free new features. An example encountered is a Dutch cash register software manufacturer, whose users typed more or less blindly into the system and checked only every ten seconds to see if the screen was showing the desired result. The simple displacing of a button in the user interface raised so many complaints (*Ccost* exceeds *Cval*) that they decided to freeze the user interface to their application in between minor releases as much as possible.

**7. Releasing too often is bad** - The aforementioned bookkeeping manufacturer started releasing on a weekly basis at some point, to shorten the feedback cycle to developers. The vendor did receive more bug reports, but product experience in general, declined. They decided that this was not caused by the fact that they released too often, but that they released to their final customers too often. The frequent releases were maintained, but only for internal use, quality assurance, and pilot customers.

**8. A quiet customer is a happy customer** - An informal survey amongst a number of customers of a Dutch plug-in software vendor showed that customers who contacted the helpdesk in the early phases of its use were much more content with the product than those customers who had not called the helpdesk in the early adoption phases. Another example encountered was a software vendor who called up a customer for a yearly check-up, and heard that they had recently decided to buy a competitors product, even while the customer still had a contract with the current vendor. This demonstrates the importance of regular customer contact. The customer would still have been a customer if the vendor had made the customer aware of the fact that *Ccost* is smaller than *Cval.*

**9. Customers (want to) read release notes** – Especially system managers of large software products are well accustomed to browsing through release notes, trying to find that one fix to a bug or that one new feature that justifies a customer's investment into updating the product. Clearly, this is a pro-active customer that is looking to optimize the value of the software product's latest release. These system managers, however, would be much more interested in information about new releases that specifically targets them. One software vendor [11] is currently experimenting with a system that filters release notes for specific customers, such that they do not receive information that is irrelevant to them. An example of this is a bug fix to a component a customer has not purchased.

**10. Having many different releases out in the field is bad** - The earlier example of the content management systems manufacturer shows us that having many different releases out in the field is not necessarily a bad thing, as long as it is part of the business model. This vendor, for instance, charges its customer for all services in the form of a service contract, especially to those customers with very old versions. To the software vendor these customers present more of a knowledge management problem, since many of the solutions built in the past have to be reused for customers experiencing similar problems now. The vendor does agree that this is only possible due to its small "manageable" amount of customers.

Much like the ten commandments [12], this list contains ten items because the author thought ten items sounded better than nine or eleven. Some other misconceptions encountered were "our next release must contain less bugs than our previous release to satisfy customers" and "we shouldn't build an automatic updater because the customer will feel they're not in control". These misconceptions are proven wrong by our cost/value functions as well, but we simply encountered them less often than the ten mentioned here. It is our firm belief that taking the profitability approach with regards to release package planning in a commercial environment is the way to go. An interesting question of validity is whether this type of anecdotal evidence is enough to prove that our cost/value functions are correct. It is part of our future work to further evaluate the validity of the cost/value functions based on historical results from both software implementations at customers and software release history.

## 4 Reducing Costs of Release Management

Besides using these functions for daily decisions, they allow us some thought experiments. Software vendors generally adhere to bug

fix/minor/major release scoping. When looking solely at version numbers, an open source project such as Mambo, has had three major releases since 2001, approximately 10 minor releases, and approximately 120 bug fix releases. These numbers show that bug fix updates are released much more often than major updates. Also, when looking at customer behaviour, they are more inclined to regularly update to a new bug fix release than they will update to a costly major release.

When looking at bug fix updates and the functions presented earlier the cost/value calculation impact factors change compared to major updates. In the case of a major update, the cost of development will largely exceed all other costs, making those less important from a financial point of view. For a major release, for instance, the completeness checking of artefacts will be a relatively small step in the release creation project. When looking at a bug fix project, however, the development might have taken only a couple of days developing effort, whereas the creation of the release package might take an equal amount of time and effort. If we then take into account that these bug fix releases generally do not generate profit and only improve product quality and reduce the amount of support calls, other costs are suddenly much more drastic.

Besides the scope of a release, the amount of customers who update to a new release determines how much effort must be put into reducing the cost of release management.
For Exact Software and its 160,000 customers, for instance, the reduction in cost by introducing a combined software configuration management system and customer relationship system was huge. By combining these two systems they enable customers to automatically download and deploy bug fix and minor updates [7]. However, if a vendor only serves twenty customers and is not planning to extend their customer base beyond one hundred customers, it must consider whether it is worth investing much into automatically releasing, delivering, and updating releases at its customers.

A product software vendor can reduce its costs in a number of areas. This cost reduction in turn enables a vendor to release more often. Releasing more often generates feedback about new releases quicker, which enables a vendor to improve its product and make better informed decisions on development and fixing plans. Cleary, this theory supports the agile camp, in its “Release early and often” viewpoint [2].

## 4.1 Vendor Side Cost Reduction

To begin with a vendor must strive to **release often**, if not continuously [17]. The more a product under development is in the shape it will be in when finally released, the less chance there is for errors to be introduced during release creation. After all, any party within the vendor organization, be it pilot customers, other developers, or the quality assurance department, will use this latest release for internal evaluation. The parties responsible for the final release will also have less work in the final stages of release creation, a process that takes place often. This process is hampered by a product that supports different languages, since quite often these language files are translated shortly before the final release date.

The process of **release creation must be automated** as much as possible to eliminate simple (error sensitive) manual tasks. If a release is checked for completeness automatically each time a release is created, it does not need to be checked extensively by quality assurance, eliminating a large part of this process.

The cost of software delivery is greatly minimized if **all delivery is done through a network** instead of expensive media, such as CDs or DVDs. The releases stored on these media are never as up-to-date as the ones stored in the vendor’s release repository, which could be accessible through a network or secure Internet connection.

## 4.2 Customer Side Cost Reduction

Whereas the vendor might be reducing costs internally, it must invest in making the deal to update to a new version as attractive as possible. Though this seems like a large investment at first, the payoff comes quickly when customers become more eager and better informed with regards to releases a vendor offers.

Software deployment costs can be reduced for the customer by **automating the update process**. This requires the software vendor to seriously invest into an update tool and to develop its architecture in such a way that customisations remain functional after an update. The main cost reduction comes from risk reduction for the customer. Even though this seems like a large investment up front, it makes the decision for a customer to update easier, and as such makes them more eager to update often. The same holds for the reduction of downtime, since customers will be much more eager to update if downtime is reduced to a minimum.

Before customers can update to a new release, however, they need to be informed about the new release. Currently, most software vendors inform their customers through information news letters, customer days, e-mails, and many other ways. A higher rate of release penetration can be reached, however, if the vendor **uses the software itself to inform the customer**. This can range from a small pop-up when the application starts up, to an automatic pull of an update, such as Mozilla’s Firefox1[1] currently does.

With regards to informing customers, release notes are an essential part of release management. When customers are looking for a bug fix, for instance, they will browse through the release notes looking for that specific piece of information. Clearly these release notes **desire a search function or link-up to the service call database**, such that customers who previously requested information concerning a problem are informed as soon as a fix for that problem has become available.

## 5 Discussion

The process of release package planning is greatly simplified with the use of the four provided functions. These functions also seem to defend that software vendors invest into automating processes

[1] http://www.firefox.org

such as release creation, release publication, informing the customer of a new release, and updating. The fact that this does not happen raises a number of questions, such as why the vendors do not invest more into these processes. An answer often given when product software vendors were confronted with this question was that they are busy creating ERP/3d-drawing/facility management/ content management/etc. software already, but that they would be happy to buy a tool that helps automating these tasks.

Part of the future work thus is to find methods and tools that assist software vendors in automating the tasks of release creation, release publication, informing the customer of a new release, and updating a customer's configuration. In earlier work the lack of tools for software deployment was identified [10] and possible solutions were presented [8]. With respect to continuous software releasing the tool Sisyphus was built to support software vendors with automatically creating their software releases [17]. Work recently has started on the Pheme prototype, a communication infrastructure that assists software vendors in sharing software, data, feedback, licenses, and commercial information with its customers.

A weakness of the cost/value functions is that being obsessed with profits will lead any product software vendor without a vision to the abyss (see Hoch et al. [5] for an example). Vendors must take into account customers will always be prepared to offer large amounts of money to small vendors if they just build one little feature that is extremely valuable to them. The vendor must always keep in mind that it is creating software for a market and not one particular customer. The functions must only be used once the prioritization of requirements for the next couple of releases has been finalized.

These calculations provide a decision method for updating and releasing, but only in case all costs and prognoses are exact. Knowing that this is impossible, we leave it to the practitioner to implement a risk factor for unforeseen costs (and unforeseen value).

## 6 Conclusion

This paper presents cost and value functions that product software vendors can use to evaluate whether it is profitable to release a version of their software. Simultaneously, functions are provided that assist an end-user or customer in making the decision to update a vendor's software product. These functions support ten changed viewpoints that were encountered in seven case studies. Finally, these functions show that costs can be saved for both product software vendors and customers on commonly occurring patch and minor updates, which can shorten feedback cycles from end-user to product software developer.

## References


[1] A. J. Bagnall, V. J. Rayward-Smith, and J. M. Whittley. The next release problem. In Information and Software Technology, volume 43, pages 883–890, 2001.

[2] K. Beck and M. Fowler. Planning Extreme Programming. Addison-Wesley, 2001.

[3] K. Bennet and V. Rajlic. A staged model for the software lifecycle. In IEEE Computer, July, 2000.

[4] P. Carlshamre. Release planning in market-driven software product development: Provoking an understanding. Springer-Verlag, 2002.

[5] D. Hoch, C. Roeding, G. Purkert, S. Lindner, and R. Muller. Secrets of software success: Management insights from 100 software firms around the world. McKinsey, 2000.

[6] S. Jansen. Software release and deployment at a content management systems vendor: a case study report. Institute of Computing and Information Sciences, Utrecht University, Technical report UU-CS-2006-0XX., 2006.

[7] S. Jansen, G. Ballintijn, S. Brinkkemper, and A. van Nieuwland. Integrated development and maintenance for the release, delivery, deployment, and customization of product software: a case study in mass-market erp software. In Journal of Software Maintenance and Evolution: Research and Practice, volume 18, pages 133–151. John Wiley & Sons, Ltd., 2006.

[8] S. Jansen and S. Brinkkemper. Modelling deployment using feature descriptions and state models for component-based software product families. In 3rd International Working Conference on Component Deployment (CD 2005), LNCS. Springer–Verlag, 2005.

[9] S. Jansen and S. Brinkkemper. Definition and validation of the key process areas of release, delivery and deployment of product software vendors: turning the ugly duckling into a swan. In proceedings of the International Conference on Software Maintenance (ICSM2006, Research track), September 2006.

[10] S. Jansen, S. Brinkkemper, and G. Ballintijn. A process framework and typology for software product updaters. In Ninth European Conference on Software Maintenance and Reengineering, pages 265–274. IEEE, 2005.

[11] S. Jansen and W. Rijsemus. Balancing total cost of ownership and cost of maintenance within a software supply network. In proceedings of the IEEE International Conference on Software Maintenance (ICSM2006, Industrial track), Philadelphia, PA, USA, September, 2006, 2006.

[12] Jeremiah, Baruch et al. Exodus 20:2-17 or Deuteronomy 5:6-21, Old Testament. In Edition used: New Revised Standard Version of the Christian Bible, 640BC-580BC.

[13] C. S. P. Ng, G. G. Gable, and T. Chan. An erp maintenance model. In 36th Hawaii International Conference on Systems Sciences (HICSS), 2003.

[14] J. N. och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell. A linguistic-engineering approach to large-scale requirements management. IEEE Software, 22(1):32–39, 2005.

[15] K. Rautiainen, C. Lassenius, J. Vahaniitty, M. Pyhajarvi, and J. Vanhanen. A tentative framework for managing software product development in small companies. In Proceedings of the 35th Hawaii International Conference on System Sciences, 2002.

[16] I. van de Weerd, S. Brinkkemper, R. Nieuwenhuis, J. Versendaal, and L. Bijlsma. A reference framework for software product management. Institute of Computing and Information Sciences, Utrecht University, Technical report UU-CS-2006-014. Submitted for publication.

[17] T. van der Storm. Continuous release and upgrade of component-based software. In Proceedings of the 12th International Workshop on Software Configuration Management (SCM–12), 2005.


## Scrum werkt… En soms ook minder…

*Marc Rissewijck, ICT*

**Scrum leeft – Scrum groeit… Waarom? Omdat Scrum werkt… En soms werkt het ook minder… Dat weten we bij ICT Embedded inmiddels na projectervaringen van afgelopen jaar. Ook weten we dat we er nog niet zijn en dat we zullen blijven aanpassen – in de ware zin van het begrip Agile.**
**Graag wil ik enkele ervaringen & lessons learned delen, die we bij ICT Embedded bij het invoeren van Scrum hebben opgedaan. Geen rocket science, wel praktisch. Niet makkelijker, wel leuker.**

### Hoe het begon…

Voor mij persoonlijk kwam ik bij een opdracht voor BenQ Mobile –toen een grote klant van ICT- voor het eerst in aanraking met de invoering van Scrum op grote schaal. Hierna heeft Ken Schwaber mij getraind tot *Scrummaster* en ben ik erg enthousiast geworden.
Niet lang hierna startte ICT een aantal projecten die relatief klein waren, en ook zeer *innovatief* –soms slechts ideeën die nog verder uitgewerkt moesten worden. Deze projecten hadden verder gemeen dat ze *snel* moesten aantonen dat de bijbehorende investeringsvoorstellen *levensvatbaar* zijn. Lastig, maar typisch geschikt voor Agile project management om zonder overbodige overhead tot snel resultaat te komen, zelfs als er nauwelijks *requirements* zijn. Daarom werden de projecten volgens Scrum uitgevoerd.

### Scrum in a nutshell

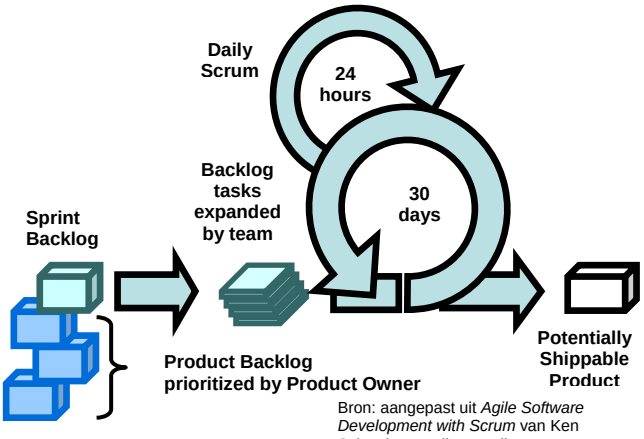
Scrum is een *lichtgewicht project management methode* en een typisch *Agile proces*. Dit laatste betekent dat mensenwerk, samenwerking en omgang met wijzigingen altijd centraal staan om werkende software te realiseren. Ondergeschikt hieraan zijn zaken als processen & tools, documentatie, contracten en vasthouden aan het plan.
Scrum geeft handen en voeten aan de projectmatige uitvoering door het zelf-organiserende ontwikkelteam centraal te stellen en zelfs volledig verantwoordelijk te stellen voor alle maandelijkse software incrementen. Dit gebeurt op basis van requirements die iedere maand zouden kunnen veranderen door de *Product Owner*. Deze ene persoon vertegenwoordigt de klant en andere stakeholders tezamen en bepaalt naast de requirements ook de prioriteit hiervan.
Nu lijkt dit alles misschien een vrijbrief om gezellig samen te hacken aan een stuk code, maar dat is bij Scrum absoluut niet het geval. Integendeel! Scrum zelf is een zeer gedisciplineerd proces met een aantal eenvoudige regels die echter erg strikt gehandhaafd moeten worden. Dit doet de *Scrummaster* die verder verantwoordelijk is voor de productiviteit van het Team door blokkerende issues op te pakken, iets wat normaal een projectleider ook doet.

Het Scrum proces is simpel en gebaseerd op korte iteraties, *sprints* genaamd, vaak van 30 dagen. Bij het begin van de sprint toont de *Product Owner* zijn productwensen en bepaalt het Team welke hiervan ze aan het eind van de maand hebben omgezet in een werkend product. Hiervoor maken ze een teamplanning en voeren deze simpelweg uit. Om de voortgang te bewaken heeft het team iedere dag een kort overleg (15 minuten) en tot het eind van de sprint wordt iedere verandering (in requirements, team, …) buiten de Team-deur gehouden. Zo kunnen ze ongestoord en gefocusseerd ontwikkelen. Let wel: ontwikkeling binnen een sprint betekent hier zowel ontwerp, code als test om te komen tot een demonstreerbaar product te aan het eind van de sprint. Dan wordt namelijk de gerealiseerde functionaliteit getoond aan de *Product Owner* en stakeholders, en zien zij dus ook wat niet af is. Op basis hiervan kunnen zij bepalen wat ze voor de volgende sprint belangrijk vinden. Verder evalueert het Team de afgelopen sprint om afspraken ter verbetering van de volgende sprint te maken. De volgende sprint start meestal de dag nadat de vorige is afgelopen en zo begint alles van voren af aan.

Omdat Scrum duidelijk een ander proces is dan de traditionele ICT projectaanpak (vergelijkbaar met Prince 2), hebben we bij onze Scrum-projecten dan ook een aantal in het oog springende facetten ervaren die tegelijkertijd ook wel “open deuren” zijn. Hierbij valt vooral de grote *gedrevenheid* van het team op hoewel die in moeilijke situaties soms ook erg moest groeien. Hiermee samenhangend blijkt het teamwerk *intenser* en raken de mensen meer *betrokken*; het werk is *leuker*. Dit, in combinatie met een scherpe focus op een steeds werkend resultaat leidt duidelijk tot *hoge productiviteit*.

Aan de hand van een aantal thema’s zal ik hieronder een indruk geven wat Scrum nu voor onze projecten betekent.

## Flexibiliteit




Bron: aangepast uit *Agile Software Development with Scrum* van Ken Schwaber & Mike Beedle.

Figuur 1: het Scrum proces

Flexibiliteit begint al bij de inrichting van Scrum zelf in het project. Voor 1 project is bijv. gekozen om niet 'full'-Scrum te gaan, maar op basis van de standaard aanpak een paar componenten van Scrum mee te nemen, zoals de daily scrum meetings en sprint-gebaseerde opleveringen. Gaandeweg is hier teamplanning en eigen verantwoordelijkheid bijgekomen. Andere projecten hebben Scrum volgens het boekje gevolgd, waarbij iedere *Scrummaster* en ieder Team hier toch ook weer een eigen invulling aan geeft. Met name de steeds terugkerende sprintevaluatie geeft het Team eigenlijk een permanente sturing. Zo hebben teams bijvoorbeeld besloten zichzelf voor een bepaalde sprint op te splitsen maar ook juist omgekeerd zich weer samen te voegen. Dit gebeurde altijd in het belang van het product en de *Product Owner*, en altijd zonder veel overhead.
Verder is het vanwege productiviteit essentieel dat de productwensen niet wijzigen gedurende een sprint. De duur van een sprint hoort niet langer te zijn dan de periode waarin de requirements bevroren kunnen blijven. Bij ICT duurden de sprints meestal van 3 weken.

### Verantwoordelijkheid

Aan de andere kant is flexibiliteit ver te zoeken als het gaat om de scheiding tussen verantwoordelijkheden: de 'wat (eerst)?'-vraag beantwoordt de *Product Owner*, het Team bepaalt het '(hoe te) doen?' en de *Scrummaster* bewaakt dit vraag-en-antwoord-spel erg strikt en zorgt voor een ongestoord en gefocusseerd team. Deze strikte scheiding wordt gewaarborgd door de directe en open communicatie die ook in Scrum zit ingebouwd: je spreekt elkaar aan op verantwoordelijkheid en bijdrage, direct. Dit geeft duidelijkheid en transparantie, terwijl het vaak ook over lastige zaken gaat. Het beestje wordt bij zijn naam genoemd, wat soms ook best confronterend kan zijn zoals een Teamlid dat een ander lid aansprak: "Je hebt de afgelopen 3 dagen nu wel dit-en-dat uitgezocht, maar wat is eigenlijk jouw bijdrage voor de sprint behalve ons lastig vallen? Over een week is de sprint review en moeten we al leveren." *Peer-pressure* in een omgeving waar dit ook kan. Dit vergt gewenning, vooral als de teamleden uit een traditionele omgeving komen. Het is belangrijk dat de *Scrummaster* of de *Scrumcoach* hier aandacht voor hebben.

### Creativiteit

Een andere vorm van flexibiliteit –of beter: creativiteit– kwam naar voren in een Team die in hun sprint een user interface op een display van gloednieuwe hardware moest tonen. Helaas bleek onze partner de display driver hiervoor niet volgens afspraak te kunnen leveren. Wat nu? De *Scrummaster* belde de partner natuurlijk om de dag, maar ook het Team keek nog even verder. Iemand kende nog wel een concullega van een project bij een klant waar met een vergelijkbaar display was ontwikkeld. Helaas, hiermee kwam hij niet veel verder behalve dat hij nu wel via internet-forums contact kon maken met de originele hardware ontwikkelaar van het display. Deze bleek nog wel ruwe testcode te hebben, en mailde dit wel even. Het was dan ook erg mooi om bij de *sprintreview* de foto's van de hoofden van de Teamleden op het display te zien!

### Vrijheid, blijheid?

Zelf-organiserende teams met verantwoordelijkheid en *Scrummaster* juist zonder: kan dit wel goed gaan? Al bij de eerste sprintevaluatie bleek van wel: hier gaf de *Scrummaster*, een ervaren projectleider, expliciet aan dat hij duidelijk het gevoel had meer controle te hebben over het project. Door de korte *daily scrums* was hij dichter bij de echte zaken die binnen het Team speelden: veel concreter en directer. Opkomende zaken werden automatisch door het Team opgepakt. Paradoxaal genoeg gebeurde dit zonder dat hij hier achteraan hoefde te gaan of te controleren.
Als vrijheid echter toch teveel overgaat in blijheid en het Team niet zelf ingrijpt, zal de *Scrummaster* dit doen, met name omdat de productiviteit in het geding is. De *Scrummaster* houdt dan meestal het Team een heldere spiegel voor in relatie met de situatie en verantwoordelijkheden.

### Projectleider-gevoel

In de projecten ervaren zowel *Scrummaster* als Team een "minder projectleider-gevoel" in vergelijking met "normale" projecten. Beide ervaren dit over het algemeen ook juist als prettig. De *Scrummaster* hoeft zich minder als traditionele projectleider te gedragen, dus minder planning, minder controle, minder verantwoordelijkheid. Het Team ziet ook minder een projectleider: niet iemand die gedurende een sprint sturing en controle uitoefent of zich

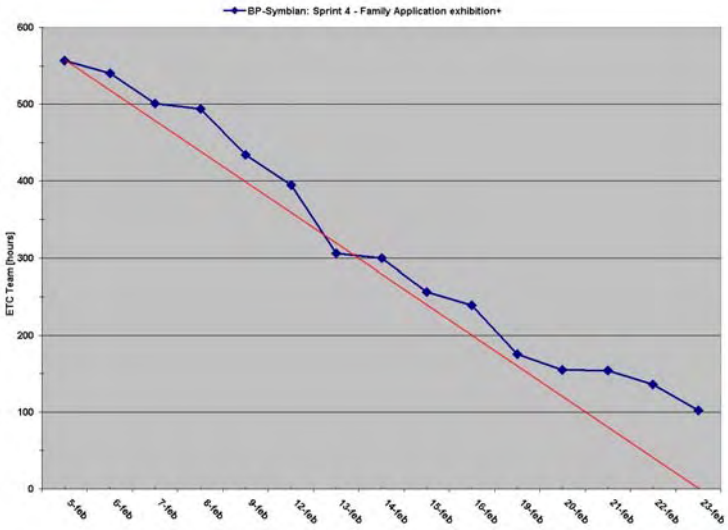
ertegenaan bemoeid. Dit blijkt meestal ook niet wenselijk of noodzakelijk omdat het Team duidelijk zelf zijn verantwoordelijkheid beleeft en zich hiernaar gedraagt.

### Teamgevoel

Voor dit “mindere projectleidersgevoel” komt een “meer teamgevoel” voor terug, tenminste als het goed is. Dit is ook noodzakelijk omdat bij Scrum het Team de absolute motor van het project is.
Des te minder teamgevoel, des te minder verantwoordelijkheidsgevoel. Zo was er een Team dat vooral bestond uit mensen die voor 50% aan het project deelnamen. De mensen zaten er half in en kwamen aanvankelijk niet op gang door gebrek aan organisatie en communicatie en dus ook zelfsturing. Vergelijkbaar hiermee is het niet optimaal voor hecht teamwerk om leden te spreiden over te veel kamers en zeker niet over meerdere ontwikkelcentra in het land. Gelukkig komt dit in de sprintevaluaties natuurlijk naar boven en pakt het Team zo goed als mogelijk verder op.
Zo is ook pro-activiteit en flexibiliteit vereist van het Team en eigenlijk ieder teamlid, om de teamverantwoordelijkheid waar te maken. Aangezien iedereen hier natuurlijk verschillend mee omgaat en voor Scrum dit absoluut essentieel is, vraagt dit soms expliciet om aandacht.

### Focus en productiviteit

Aan de andere kant: een gedreven en hecht Team kan kleine wonderen verrichten en is erg productief




[Figuur 2: een Burndown-chart]

als de centrale focus op een duidelijk sprintdoel is gezet. Alle teamactiviteiten worden hier dan direct of indirect op gericht. Tijdens evaluaties gaven teamleden dan ook aan dat ze gedurende de sprint veel meer gefocussed waren op hun werk, op de taken van die dag. En omdat het Team zelf de taken bepaalde om de sprintdoelen te halen, leefde de planning in het Team en voelde men ook hoe de taken onderling samenhingen, ook wat betreft prioriteit. Zo komt het geregeld voor dat teamleden elkaars taken overnemen om te komen tot het beste sprint resultaat. Echt teamwork. Al met al leidt deze gezamenlijke focus tot hoge productiviteit, waarbij Teams soms expliciet stelden dat de resultaten niet mogelijk zouden zijn in een “normale” projectvoering.

## Burndown-gevoel

Met deze focus samenhangend gebruikt Scrum een zogenaamde *burndown-chart* waarbij alle Teamtaken dagelijks geupdate worden met betrekking tot nog te “verbranden” Teamuren voor de sprint. Deze *burndown*-lijn hoort ongeveer lineair naar beneden lopen en bij het einde op nul staan. De combinatie van deze grafiek met de daily scrums is erg bruikbaar en krachtig om voortgang te bespreken. Wel is door een enkeling duidelijk een negatieve druk gevoeld om taken af te ronden. Iedere dag word je binnen het Team met de neus op de feiten gedrukt, en is de stand van zaken (te) duidelijk.

**Dus...**

Zoals altijd is ieder project anders, en dat maakt het ook leuk. Gezien het karakter van Scrum, zijn Scrum-projecten nog wat meer anders, wat het eigenlijk ook nog leuker maakt. De ervaringen zijn gevarieerder en dat maakt het moeilijker conclusies te trekken. Er is simpelweg meer mogelijk om de projectdoelen te halen.

Slechts enkele onderdelen van het Scrum proces zijn nieuw. De echte kracht van Scrum zit dan ook niet in 1 of 2 van deze componenten, maar vooral in de combinatie van alle componenten. Het een versterkt het ander en omgekeerd, en gezamenlijk leidt dit tot een krachtig draaiende motor die zichzelf continu aanpast om tot betere resultaten te komen. Mensenwerk, samenwerking en flexibiliteit staan hierbij centraal, en op deze manier hebben wij bij ICT (zeer) positieve ervaringen met Scrum. Gewoon doen!

Inmiddels leggen we nu ook focus op nieuwe aandachtspunten die goed bij Scrum aansluiten: *Continuous Integration* en *Test Driven Development*. Deze Agile processen dragen vooral bij aan continu werkende software en teamwerk. En dat hopen we natuurlijk ook de komende periode in concrete projecten te ervaren!

*Marc Rissewijck (marc.rissewijck@ict.nl) is Consultant, Certified Scrum Master en Scrum Coach bij ICT Embedded.*

## Terugblik Q conferentie 9 mei 2007

*Martin Muller, voorzitter Q-society, bestuurslid SPIder*

Op 9 mei jl. werd opnieuw een conferentie georganiseerd door de “Society for Quality Professionals in ICT” de derde event sinds de oprichting van de Q Society in 2005. In dit artikel wordt ingegaan op dit zeer geslaagde event.

De locatie was deze keer het Aluminium Centrum in Houten. Er waren verschillende sprekers uitgenodigd om op een interactieve wijze met het publiek hun visies en stellingen te formuleren – en met het publiek te delen – rondom het thema: *“Kwaliteit! Tot hier en niet verder!”* Deze titel werd gekozen om opnieuw een discussie los te maken rondom kwaliteit door practitioners in kwaliteit maar ook afnemers van kwaliteit(producten in ICT) te stimuleren om actief met elkaar over kwaliteit in discussie te gaan. Gezien de vele reacties uit het publiek zijn we er in geslaagd om de nodige pittige discussies los te trekken! In totaal waren met de organisatoren en presentatoren meegerekend, precies 100 mensen aanwezig.

### *De inleiders*

1. Is ‘goed genoeg’ wel goed genoeg (Paul Klint, *Centrum voor Wiskunde en Informatica en Universiteit van Amsterdam*)
2. Hoe manage ik de kwaliteitsperceptie van CxO’s? (Jan van Broeck, *Threon Europe*)
3. Waar ligt de bodem bij kwaliteit? (Peter Brouwer, *Process Improvement Manager ABN Amro Business Unit Nederland)*
4. Kwaliteit en ik (Q&I) (Cees Michielsen, ITIB, *Product creation process coach)*
5. If Leonardo was a quality guru *(Jens Pas, verhalenverteller)*

### *Introductie en opening*

Als dagvoorzitter mocht ik de conferentie aftrappen. Paul Klint was de eerste plenaire spreker. Naar keuze konden de conferentie deelnemers één van de 3 parallel tracks naar keuze volgen. Na afronding van de parallel tracks werd plenair door Jens Pas een afsluitende presentatie gehouden en werd het formele gedeelte van de conferentie door mij afgesloten.

### *Korte impressies van de presentaties*

Hierbij worden de presentaties kort weergegeven. Opgemerkt moet worden dat dit een persoonlijke interpretatie van mij betreft. Gelegde accenten en nuances zijn zeker subjectief, toelichtingen en formulering daarvan zijn daarom volledig voor eigen verantwoording en zijn niet afgestemd met de auteurs.

**Is ‘goed genoeg’ wel goed genoeg (Paul Klint)**

Paul verzorgde een flitsende opening met zijn fraaie slides en zijn directe vragen aan het publiek. De problematiek van en een toelichting op de wereld van onder meer geldautomaten, Bling Bling GSM-en en MRI scanners – waarin steeds meer software is ingebouwd – volgden elkaar in rap tempo op. Zijn visies op en uitspraken over software kwaliteit omvatte bekende maar deels ook onbekende zaken. Wat te denken van perfectionisme naast elegantie en vakmanschap als we een esthetische invalshoek op kwaliteit kiezen? Is perfectionisme goed of niet goed? Of ‘achter het spook aanjagen’ zoals Paul dit noemt als we het hebben over (toekomstige) klantenvisie, die niet bekend is, continu wijzigt en daardoor vluchtig is. Dit zal zeer herkenbaar zijn voor de pakketen bouwers onder ons die immers te maken hebben met (toekomstige) klanten die (toekomstige) eisen en wensen ten aanzien van (toekomstige) producten en diensten hebben. Hoezo kwaliteit goed genoeg?
Een andere verfrissende benadering is die van een probleemgerichte kijk op kwaliteit. Paul stelt dat je de aandacht niet op kwaliteitsattributen moet richten maar op de problemen die ze op moeten lossen. Als die problemen zijn opgelost zijn volgens Paul de kwaliteitsattributen vanzelf goed genoeg. Software Quality Assurance zou dan ook naar een acceptabel risiconiveau moeten streven. Kwaliteit is dan automatisch goed genoeg.
Over de vraag hoe het beter kan gaf Paul het volgende aan:

- Investeer in mensen “Zie de software engineer als held”
- Investeer in software kennismanagement (processen, technieken en vooral de integratie daarvan)

De uitspraak “Zie de software engineer als held” is interessant en staat nogal haaks op de term die in CMM implementaties vaak wordt gebruikt: *“no more heroes”*. In Paul zijn uitspraak staan sociale en financiële waardering voorop. Zijn aanbeveling is dan ook om te (blijven) investeren in opleidingen en bijscholing. In CMM staat “no more heroes” voor het feit dat processen herhaalbaar moeten zijn en niet afhankelijk van enkele heroes.

In het tweede gedeelte van Paul zijn presentatie werd nader ingegaan op het principe en het belang van een Software Kennisbank (SKB). In de ‘ideale’ situatie: bevat de kennisbank alle kennis over de software. Van daaruit wordt het hele bouw en leveringsproces gecontroleerd. Versiebeheersing van software en herhaald en snel kunnen hertesten zijn zaken die in een SKB een belangrijke rol spelen. Bij Paul’s toelichting dat de technische mogelijkheid om snel opnieuw code te kunnen aanmaken/wijzigen ontspon zich de pittige discussie dat dit weliswaar mooi lijkt maar dat daardoor de ontwikkelaars mogelijk geneigd zijn om sneller en slordiger te werken omdat het toch allemaal wordt gecheckt en snel kan worden bijgesteld/herhaald! Dit zou dus juist niet leiden tot verbetering van kwaliteit! Beide visies zijn natuurlijk te verdedigen.

Tot slot Paul zijn conclusies:

- Risico analyse bepaalt of software “Goed genoeg” is
  - maatschappelijke en economische risico’s
- Een integrale SKB leidt tot “betere” software:
  - complete beheersing van alle softwarekennis
  - continue integratie en levering
  - analyse en transformatie van software vormen een evolutiepad naar een SKB én een mechanisme voor kwaliteitverbetering

### Hoe manage ik de kwaliteitsperceptie van CxO's? (Jan van Broeck)

In het begin van zijn verhaal nam Jan ons mee in de (kwaliteit)evolutie van de afgelopen jaren. Een citaat wat mij hierover opviel. *"Commercial pressure means that results need to be delivered in ever decreasing timeframes, with quality often losing out in a trade-off against speed te market"*. Verder leidde Jan ons bij zijn vraag "Waar zit de fout?" naar zijn 4 (mogelijke) fout oorzaken:

- Strategie
- Projecten of programma's
- Operaties
- Communicatie (link met "perceptie")

Een eerste conclusie hierover:

- Het heeft dus alles te maken met het in overeenstemming brengen van 'verwachting" en "uiteindelijk resultaat en tevredenheid".
- Product kwaliteit is vermoedelijk eenvoudiger kwantitatief te beschrijven ten opzichte van service kwaliteit
- Managers zijn gevoelig voor "klanten" …
- Benader dus elk kwaliteitsprobleem vanuit de 'finale klant' situatie

Op zijn vraag waar kwaliteitsverwachting wordt gecreëerd:

- Strategy. "Portfolio management is the bridging interface (process) between the Strategy process and Program & Project Management"
- Programs / Projects "Transition management: implementing all nonrecurring tasks necessary to achieve the desired operating model and stakeholder commitments"
- Operations "by providing feedback to Program / Projects and to Strategy"

Een paar van door Jan getrokken conclusies:

- Kwaliteit en management: Take the customer view, take a life cycle view
- Overlaad uw organisatie niet met een overvloed aan methodologieën die niet geïntegreerd zijn
- Finaal – De klant krijgt de kwaliteit die de CxO's georganiseerd hebben

### Waar ligt de bodem bij kwaliteit? (Peter Brouwer)

Peter werkt al jaren in zijn functie van PIM, Process Improvement Manager, bij ABN Amro BU Nederland. Hij deelde met ons zijn kennis en ervaring met het afkalven van kwaliteit als gevolg van een afslankingsoperatie en grootschalige reorganisatie naar een multi vendor model nadat het Inspiration programma (gecombineerde invoering van CMM en DSDM) in eerste instantie succesvol was beëindigd, en met nieuw elan geven van kwaliteit in de nieuwe opzet ná Harvest en Symphony. Hij benoemde de trits: "Forming – Storming – Norming – Performing. Harvest was dat deel van het programma dat zich richtte op outsourcing van applicatieve software terwijl Symphony het deel was dat zicht richtte op infrastructuur. Beide delen zijn inmiddels in aparte organisaties belegd. De BU NL is verantwoordelijk voor de regie over de verschillende vendors.
Peter vertelde ons over de paradox van snelheid en kwaliteit. Er ligt een spanningsveld tussen snelheid en kwaliteit, institutionalisering van QA neemt jaren in beslag en de "echte" business ziet QA niet. Tijdens de 'storm' moet QA zich aanpassen aan de veranderende wereld. Twee belangrijke trends speelden daarbij:

- aanpassen aan de capaciteit van de organisatie (QA organisatie verkleinen)
- projectmanager "beheert" de kwaliteit van een ander project (van projectmanager naar subcontract manager)

Dit vergt communiceren, communiceren, communiceren…
De bodem van kwaliteit ligt volgens Peter in het verkleinen maar wel in stand houden van de QA organisatie. En daarnaast vanuit QA procesverbetering in stand houden en QA focussen op datgene wat er echt toe doet. QA klimt weer uit het dal doordat de basis aanwezig is QA sneller wordt herkend als bijdrage tot herstel.

Lessons learned van Harvest en Symphony:

- Niets is moeilijker dan gewoonten veranderen
- Voordelen moeten aantoonbaar zijn
- Stel concrete en realistische doelstellingen en toets achteraf
- Veranderingsgezindheid is omgekeerd evenredig met de hoeveelheid doorgemaakte veranderingen
- Veranderingsgezindheid van a naar b is omgekeerd evenredig met de inspanning om naar a te komen
- Verwachtingsmanagement is cruciaal
- Verzand niet in details *Langzaam veranderen gaat altijd nog sneller dan snel veranderen!*

Conclusies: QA overleeft een zwaar veranderprogramma mits:

- Een zeker basisniveau aanwezig is
- Management Commitment continu aanwezig is
- QA zich aanpast aan de veranderende wereld
- QA focust op waardetoevoeging
- De QA bodem op het netvlies blijft

E.W. Dijkstra

### Kwaliteit en ik (Q&I) (Cees Michielsen)

In de presentatie stond het individu in het voortbrengingsproces centraal. Een aantal stellingen stonden haaks op die van Paul Klint uit de openingssessie. Met name de kwaliteiten en het vakmanschap van de software engineer kwamen hierbij aan de orde. Als voorbeeld hiervan werd Edsger Dijkstra genoemd, die al in de jaren zestig en zeventig (vorige eeuw!) een methode heeft ontwikkeld waarmee de correctheid van software

algoritmes aangetoond kan worden. Daarmee wordt de kwaliteit van het product geen toevalligheid meer, maar eerder het voorspelbare resultaat van een gedisciplineerd proces. Echter, hoe komt het dan toch dat, ondanks de bestaande methoden, software engineers er voor kiezen om een foutgevoelige en onzekere weg te kiezen om producten te ontwikkelen?
Met een aantal voorbeelden toont Cees aan dat er meerdere factoren een rol spelen, die er uiteindelijk voor zorgen dat individuen een keuze maken en een ander gedrag gaan vertonen:

***Willen + Kunnen + Moeten + Durven = Doen***

*Willen* betreft het 'what is in it for me?' aspect. *Kunnen* gaat over de kennis en kunde die aanwezig moeten zijn. *Moeten* gaat in op de druk van de omgeving (de collega's, de baas, de organisatie) om iets van iemand gedaan te krijgen. *Durven* tenslotte refereert aan een belangrijk en vaak slecht zichtbaar aspect, waardoor mensen uiteindelijk toch niet de stap zetten. De mix van deze ingrediënten is voor elk individu anders, bij de één werkt druk van bovenaf al in voldoende mate om mensen in beweging te krijgen (zie ook bepaalde culturen: Duitse, Franse, Indische), bij ons is vaak het *willen* de sleutel. In het geval van software ontwikkeling moeten we ons de vraag stellen: willen we wel de kwaliteit leveren die we kunnen leveren? In hoeverre bindt de software discipline zich eigenhandig de strop om de nek en diskwalificeert zichzelf als engineering discipline?

**If Leonardo was a quality guru (Jens Pas)**
Specifiek op verzoek van Jens Pas moest ik hem voorstellen als 'verhalenverteller'. Hij was door ons als afsluitende key note gepositioneerd als visionair over kwaliteit en waar kwaliteit zich in de toekomst naar toe beweegt. Nou verhalen vertellen kan hij! In zijn eentje zou hij uren van ons programma kunnen vullen! Zijn visualisaties over Da Vinci zijn dan ook een lust voor het oog.
De link van Leonardo naar kwaliteit is snel gelegd. Jens onderscheidt bij het beleven van kwaliteit emotional en rational. Zijn advies is dat we meer moeten voelen (emotional dus) in plaats van schrijven, formaliseren, opleggen, controleren waar QA zich nog wel eens aan te buiten gaat.

Jens zijn afsluitende slide is typerend:
De fuik dient om de vis te vangen. Als de vis gevangen is, vergeet dan de fuik.
De strik dient om de haas te vangen. Als de haas gevangen is, vergeet dan de strik.
Het woord dient om de gedachte te vangen. Als de gedachte gevangen is, vergeet dan het woord.

***Afsluiting***
De conferentie werd afgesloten met een gratis lunch en een gelegenheid tot netwerken. De sfeer tijdens de presentaties en daarna was ongedwongen en zeer plezierig. Met 100 mensen die aanwezig waren kan je zeggen dat dit een geslaagde dag was! Met de vertegenwoordigers van de 10 organiserende bedrijven (ook ps_testware en Qualityhouse hebben zich inmiddels bij de Q Society aangesloten!) kon worden doorgediscussieerd over de conferentie, de organisatie van de Q Society, onze vervolgplannen en tal van andere onderwerpen waarin kwaliteit centraal staat. De presentaties zullen nog op de website worden geplaatst.

De Q Society is al weer begonnen met het treffen van voorbereidingen van de najaarsconferentie, gepland te houden ergens eind november van dit jaar. Via de SPIder Koerier wordt u op de hoogte gehouden van onze plannen. Hebt u ideeën, weet u thema's en onderwerpen of kent u sprekers die ons mogelijk ten dienst willen zijn, laat het ons dan zo spoedig mogelijk weten. We kunnen dan met en dank zij uw feedback, nòg beter een programma naar uw wens samenstellen!

Ik sluit af om alle deelnemers te bedanken voor uw belangstelling in ons programma. Namens de Q Society wensen wij u alvast een fijne vakantie toe. Wij komen graag in het najaar bij u terug!

*Martin Muller*
*Voorzitter Q Society, bestuurslid SPIder*

## ■ SPIder werkgroepen

### Werkgroep SPI in kleine organisaties

Per 1 mei j.l. heeft de SPIder-werkgroep "SPI in Kleine Organisaties" zich opgeheven. De werkgroep heeft zo'n 10 jaar bestaan, en in die tijd het "vijf-dagen-model" en de "SPI-starterkit" naar buiten gebracht. Ook heeft de groep tweemaal een plenaire sessie verzorgd. Uiteraard waren er ook de bijeenkomsten, naar schatting zijn dat er toch wel een stuk of vijftig geweest.
De laatste tijd bleek het steeds lastiger om een voldoend aantal deelnemers bijeen te krijgen om bijeenkomsten te rechtvaardigen. Verbreding van de scope – zo stond het onderwerp "menselijk gedrag in relatie tot process improvement en beslissingen daarover" op de agenda – mocht niet baten. Wellicht dat ook de aantrekkende economie hier een rol speelt.
Liever dan te hopen op betere tijden (sic) of voortmodderen heeft de werkgroep verkozen zich op te heffen. De groep kijkt terug op tal van zeer

levendige discussies (als ik een klein beetje overdrijf zijn we bv. rond requirements management welhaast met elkaar op de vuist geweest) en een netwerk dat tal van zakelijke relaties opgeleverd heeft. Steeds weer bleken vrijwel alle deelnemers aan de bijeenkomsten tot nieuwe inzichten te komen, en die ook in hun praktijk toe te kunnen passen. Deelnemen aan een werkgroep valt kortom iedereen die wel eens buiten zijn dagelijkse omgeving over zijn vak wil praten, van harte aan te raden.
Tot slot wil ik iedereen bedanken die bijgedragen heeft aan de werkgroep, in het bijzonder Ger Fischer voor de oprichting (10 jaar is op zich al een meer dan florissante levensduur voor een werkgroep en getuigt daarmee van een goede visie), het SPIder-bestuur voor het faciliteren van de werkgroep, en vooral al degenen die voordrachten gehouden hebben, discussies aangezwengeld, visies gedeeld, boeken gelezen en doorgegeven, ruimte en broodjes beschikbaar gesteld, of secretariaatswerk verricht hebben (Marie-Louise, dank!).

Als er ondanks de opheffing vragen zijn, neem dan contact op met Herman Rave, 0653 231 662, herman@raveonline.nl

## ■ Deelname in SPIder

Indien u actief wilt participeren in SPIder en de Koerier in de toekomst wilt ontvangen, kunt u zich aanmelden als deelnemer in SPIder bij:

**Secretariaat Stichting SPIder**
p/a Cantrijn Secretariaten
Postbus 2047, 4200 BA GORINCHEM
Tel: 0183 - 62 00 66, fax: 0183 - 62 16 01
E-mail: info@st-SPIder.nl

Aanmelding kan ook via het aanmeldingsformulier op de website van SPIder: www.st-SPIder.nl.

## ■ Nieuwsberichten & evenementenkalender

De evenementenkalender bevat een overzicht van internationale conferenties op het gebied van SPI, metrieken en softwareproductkwaliteit. Daarnaast zijn de activiteiten van SPIder opgenomen.

Ook nationale evenementen op het gebied van softwareproduct- en procesverbetering kunnen in deze evenementenkalender worden opgenomen. Via de SPIder Koerier kan een organisator van SPI gerelateerde evenementen een selecte groep van geïnteresseerden bereiken. Voor commerciële evenementen zoals conferenties, workshops, lezingen en andersoortige bijeenkomsten vraagt de redactie een kleine bijdrage in de kosten.

Ⓢ = SPIder event
✓ = korting voor SPIder donateurs

| | | | |
|---|---|---|---|
| | **2007** | | |
| Ⓢ | 6 juni | 1e bijeenkomst werkgroep requirements | |
| | 11 juni | ESEPG Conference www.espi.org | ✓ |
| | 20 juni | Bits&Chips Testdag | |
| | 2-4 juli | Profes www.liis.lv/profes2007 | |
| Ⓢ | 19 sept. | Werkgroep SPI in kleine organisaties; Menselijk gedrag, op basis van het boek "Emotie-economie" van Henriette Prast herman@raveonline.nl | |
| | 20 sept. | Testnet najaarsevent | |
| Ⓢ | 25 sept. | Plenaire sessie: Software Process Innovations | |
| | 26-28 sept. | EuroSPI www.eurospi.net | ✓ |
| Ⓢ | 2 okt. | 10e SPIder Conferentie "Energyzing Improvements" | ✓ |
| | 17 okt. | Bits&Chips Embedded Systemen | |
| | 17 okt. | Testnet Testen van pakketten | |
| Ⓢ | 7 nov. | Werkgroep SPI in kleine organisaties; Risicomanagement en Time-line planning herman@raveonline.nl | |

## ■ Colofon


De SPIder redactie bestaat uit:
Cees Michielsen en Cantrijn Secretariaten.

Voor reacties en vragen m.b.t. de ***SPIder Koerier*** kunt u zich wenden tot:
Redactie SPIder Koerier
E-mail: koerier@st-SPIder.nl

Indien u in de toekomst een herinneringsbericht wilt ontvangen over de datum van kopijsluiting, stuur dan een e-mail "opname SPIder copylijst" naar Koerier@st-SPIder.nl.

Informatie over SPIder is te vinden op de website: www.st-SPIder.nl.

Voor reacties en bijdragen op de ***SPIder website*** kunt u zich richten tot:
Redactie SPIder web, Niels Malotaux
E-mail: niels@malotaux.nl

Deze koerier kwam tot stand met medewerking van:
- ITIB (www.itib.net)
- Cantrijn Secretariaten


*We weten het niet*

[Jens Pas]